



Escola Superior de Tecnologia e Gestão

Instituto Politécnico da Guarda

DeepPi

Deteção incêndios através de Deep Neural Learning

Relatório de Projeto Aplicado submetido como requisito parcial para obtenção do
grau de Mestre em Computação Móvel

Orientador: Professor Carlos Carreto

Coorientador: Professor Adérito Acaso

Diogo Manuel de Ascensão Almeida Aleixo

03 | 2016

If i have seen further than others, it is by standing upon the shoulders of giants

[Isaac Newton]

Agradecimentos

O trabalho apresentado no presente relatório só foi possível graças à colaboração e apoio de algumas pessoas, às quais não posso deixar de prestar o meu reconhecimento.

À minha namorada por sempre me ter apoiado incondicionalmente em todas as decisões da minha vida, tendo sido nesta fase um elemento essencial para o término do trabalho.

Agradeço também ao Adrian Rosebrock pelo excelente curso sobre visão computacional. Este curso foi sem dúvida a maior valia para a minha aprendizagem no tema e serviu de apoio para construir o software. Aos autores do curso *cs231n* disponibilizado por *Stanford* e ao Michael Nielsen pelo excelente livro *Open-Soure* disponibilizado.

À minha mãe e irmãos por me terem dado a força e os ensinamentos necessários durante toda a vida para cumprir com sucesso esta etapa.

Ao meu filho que apesar da tenra idade foi um elemento essencial onde arranjei a motivação necessária para acabar o projeto para obtenção do grau de Mestre em computação móvel.

Por fim, e não menos importante, a todos os meus amigos que me acompanharam neste processo e o influenciaram direta ou indiretamente.

Resumo

O fogo é um desastre ambiental e/ou humano que pode atacar em qualquer lugar e ser bastante destrutivo, consumindo bens materiais, ambientais e humanos. Métodos existem para detetar fumo e fogo e permitir as autoridades atuar de modo a evitar ou minimizar os danos. Os métodos existentes vão desde a utilização de sensores do meio ambiente, à utilização de camaras, sejam elas de *CCTV(Closed circuit television)* ou projetadas para a deteção do fogo. O objetivo deste trabalho foi, a aplicação de algoritmos de processamento de imagem e *machine learning* para a deteção de fogo bem como o desenvolvimento de um módulo de energia renovável através de painéis fotovoltaicos.

Foi também objetivo o ajustamento do objetivo principal a Hardware de baixo custo de modo a reduzir o preço final do sistema.

Palavras-chave: Visão computacional, aprendizagem máquina, processamento de imagem, inteligência artificial , energia renovável, painéis fotovoltaicos.

Abstract

Fire is an environmental and / or human disaster that can strike anywhere and be very destructive , material, environmental and human assets consuming. Methods exist to detect smoke and fire and allow the authorities to act in order to avoid or minimize damage. Existing methods range from the use of environmental sensors, the use of cameras , whether or CCTV (*Closed circuit television*) designed for the detection of fire. The objectives of this work are the application of image processing algorithms and machine learning algorithms for fire detection and the development of a renewable energy module

It is also aimed at the adjustment of the main objective of low cost hardware to reduce the final price of the system.

Keywords computer vision, machine learning , image processing , artificial intelligence , renewable energy, photovoltaic panels

Índice

1	INTRODUÇÃO	Error! Bookmark not defined.
1.1	Enquadramento e motivação do projeto	12
1.2	Definição do problema e objetivos	12
1.3	Solução proposta para resolver o problema	13
1.4	Contribuição do projeto para o estado da arte.....	13
1.5	Organização do relatório	14
2	Trabalhos relacionados.....	14
2.1	Exemplos de trabalhos semelhantes	15
2.1.1	Métodos baseados em Bayesian Networks	15
2.1.2	Método baseados em máquinas vetoriais.....	18
2.1.3	Métodos baseados em redes neurais	21
2.2	Sistemas comerciais	23
2.2.1	Firewatch.....	24
2.2.2	ZeroFires	24
3	Tecnologias e áreas estudadas.....	25
3.1	Processamento de imagem	26
3.1.1	OpenCv	27
3.1.2	Aprendizagem máquina	28
3.1.3	Support Vector Machines.....	29
1.1.2.	Decision trees e Random Forests	32
3.1.4	Aprendizagem Ensenble	33
3.1.5	Convolutional Neural Netwoks.....	34
3.2	Energias renováveis.....	40
3.2.1	Energia solar	40
3.2.2	Energia eólica.....	Error! Bookmark not defined.
4	Análise de requisitos	41
5	Sistema desenvolvido	43
5.1	Arquitetura do sistema desenvolvido	43

5.2 Módulos do Sistema.....	44
5.2.1 Implementação do dataset	44
5.2.2 Firebase e Google Drive	54
5.2.3 Classificador.....	57
5.2.4 Serviço de streamming.....	67
5.2.5 Aplicação Móvel	68
5.2.6 Módulo Pan&Tilt	79
5.2.7 Módulo de energias renováveis.....	80
6 Testes e resultados.....	82
6.1 Testes do classificador	82
7 Conclusões.....	Error! Bookmark not defined.
Bibliografia	95

Índice de Figuras

Figura 1 - Bayesian networks.	15
Figura 2 - Bayesian Networks, dataset.	16
Figura 3 - Bayesian Networks, comparação com outros métodos (Verdadeiros positivos).	17
Figura 4 - Bayesian Networks, comparação com outros métodos (Falsos positivos).	17
Figura 5 - Bayesian Networks, comparação com outros métodos (Rapidez).	18
Figura 6 - SVM, arquitetura da solução.	18
Figura 7 - SVM, algoritmo.	19
Figura 8 - SVM, dataset.	20
Figura 9 - SVM, resultados do método de Toreyin.	20
Figura 10 - SVM, comparação com o método de Toreyin.	21
Figura 11 - Redes neurais, segmentação por cor.	22
Figura 12 - Redes neurais, escolha do tamanho da camada oculta.	22
Figura 13 - Redes neurais, algoritmo.	23
Figura 14 - Redes neurais, treino da rede.	23
Figura 15 - Redes neurais, resultados dos testes.	23
Figura 16 - Firewatch sensores.	24
Figura 17 - ZF-WildFire station series.	25
Figura 18 - ZF-Portable system.	25
Figura 19 - ZF-Dashboard.	25
Figura 20 - Logotipo openCv e python.	26
Figura 21 - Imagem do ponto de vista do computador.	27
Figura 22 - Divisão do dataset.	29
Figura 23 - Separação Linear possível.	30
Figura 24 - Separação linear impossível.	30
Figura 25 - Linha fronteira.	31
Figura 26 - Support Vectors.	31
Figura 27 - Arvore de decisão RGB.	32
Figura 28 - Random forest.	33
Figura 29 – Perceptron.	35
Figura 30 - Arquitetura de rede neuronal.	35

Figura 31 - Soma de dois bits com NAND's - Ponto de vista digital.....	36
Figura 32 - Soma de dois bits com NAND's - Ponto de vista neuronal.....	36
Figura 33 - Função de passo.....	37
Figura 34 - Função sigmoid.....	37
Figura 35 - Rede neuronal.....	38
Figura 36 - Rede neuronal VS rede convulsional.....	39
Figura 37 - Demonstração das camadas da CNN.....	40
Figura 38 - Arquitetura de hardware.....	43
Figura 39 - Processo de tratamento do dataset.....	45
Figura 40 - Fluxograma de obtenção de imagens.....	46
Figura 41 - Output script de download de imagens.....	47
Figura 42 - Output script de download de imagens.....	47
Figura 43 - Exemplo de mau input para o dataset.....	48
Figura 44 - Output script de redimensionamento das imagens.....	49
Figura 45 - Histogramas diferentes.....	50
Figura 46 - Histogramas iguais.....	51
Figura 47 - Output script elimina duplicados.....	53
Figura 48 - Output script renomear imagens.....	53
Figura 49 - Exemplo de data augmentation.....	54
Figura 50 - Arquitetura Google Firebase.....	55
Figura 51 - Diagrama de integração do sistema com o firebase.....	56
Figura 52 - Arquitetura treino remoto.....	56
Figura 53 - Arquitetura de rede do sistema.....	58
Figura 54 - Primeiro bloco.....	59
Figura 55 - Segundo bloco.....	60
Figura 56 - Bloco final.....	61
Figura 57 – Classificador.....	61
Figura 58 - Divisão do dataset.....	63
Figura 59 - Passos para carregar as imagens.....	64
Figura 60 - Diagrama funcionamento streamming.....	68
Figura 61 - Ecra de login.....	69
Figura 62 - Email mal formatado.....	70

Figura 63 - campos vazios.	71
Figura 64 - chamada de autenticação.....	72
Figura 65 - tentativa de reset da password falhada.	73
Figura 66 - Ecra para inserir nova password.	74
Figura 67 - Chamada para reset da password.	75
Figura 68 - Ecra de streamming do video.	76
Figura 69 - Notificação push em foreground.	77
Figura 70 - Notificação push em background.	78
Figura 71 - Historico das notificações de fogo.	79
Figura 72 - Esquematico ligação servo motores.	80
Figura 73 - Consumo familia raspberry pi. Adaptada de [11]	81
Figura 74 - Arquitetura LeNet.	83
Figura 75 - Arquitetura personalizada.	84
Figura 76 - Arquitetura Karpathy.	85
Figura 77 - Arquitetura mini vgg.	86
Figura 78 - PERDA EM TREINO E FASE DE VALIDAÇÃO.....	89
Figura 79 - Exatidão em treino e fase de validação.	90
Figura 80 - Previsões no dataste de teste.	91

Índice de tabelas

Tabela 1 - Matriz de confusão	87
Tabela 2 - Precisão, recall e F-score	89

Índice de algoritmos

Algoritmo 1 - Comparaçao de imagens	52
Algoritmo 2 - Criação de matriz de representação de imagens	65
Algoritmo 3 - Criação dos datasets de treino, validação e test	66

Siglas

CCTV	Closed circuit television
CSV	Comma separated values
CO2	Dioxido de carbono
FPS	Frames per secound
HSV	Hue Saturation value
LIDAR	Light detection and ranging
MMS	Multi media message
MJPEG	Motion jpeg
RGB	Red green blue
SMS	Short message system
SVM	Support vector machines

1 Introdução

Este capítulo tem como objetivo fazer uma introdução ao projeto e contextualizar o mesmo no âmbito da engenharia informática e a sua motivação. Também neste capítulo é descrito o problema a resolver, a solução proposta para a sua resolução e contribuição para projetos semelhantes. Por ultimo é exposta a organização dos capítulos que constituem o relatório

1.1 Enquadramento e motivação do projeto

Neste relatório é descrito um sistema para deteção de fogos. Este sistema é enquadrado na área técnica do processamento de imagem, inteligência artificial, aprendizagem máquina e energias renováveis. A principal motivação deste projeto é, como será descrito no próximo ponto do atual capítulo, a crescente destruição de recursos naturais e humanos pelos fogos em todo o mundo. Este facto torna urgente uma solução capaz de solucionar de alguma forma este problema. Um fogo pode causar mortes por inalação, queimaduras de vários graus (levando no caso mais extremo à morte), consumir vários hectares de terreno numa questão de horas, consumir bens materiais, entre outras coisas.

Nesse sentido, este projeto tem como objetivo ajudar os bravos bombeiros e pessoas a quem afeta os fogos afetam direta ou indiretamente.

1.2 Definição do problema e objetivos

Em Portugal e em todo o mundo, os incêndios consomem todos os anos vários milhares de hectares. Motivados pela ganância ou por um simples descuido, estes tendem a consumir áreas verdes e casas de população da zona tornando-se um grande perigo para a população em geral. Os bravos bombeiros, consomem muitos recursos e por vezes infelizmente vidas no combate deste desastre. Hoje em dia os incêndios são extinguidos com recurso a meios terrestres e aéreos que saem também eles muito caros aos cofres do estado de qualquer nação. A deteção do incêndio é atualmente muito primitiva, limitando-se a equipas estrategicamente posicionadas na floresta, a sistemas de sensores distribuídos e a visão computacional. O método de equipas na floresta, sendo humano é falível. O humano, por natureza, distrai-se com facilidade quando deparado com situações monótonas. Os sensores distribuídos pela floresta têm o problema de se limitarem a fazer a função de sensor, e “dispararem” somente quando existe e é detetado um valor anormal de CO_2 (Dióxido de carbono) por exemplo, o que pode ser tarde alem de o resultado ser influenciado pela direção do vento. Os sensores têm também o problema de o diâmetro de atuação ser limitado, obrigando a um investimento muito grande para uma área considerável. As câmaras com poder de visão computacional têm o problema do raio de visão, que apesar de poder ser

grande, pode ser influenciado com uma arvore grande, ou um rochedo. Para evitar tais problemas, estas são muitas vezes colocadas em cumes e a analisar a encosta adjacente. O problema desta solução é que embora resolva o problema em grande parte, continuam a haver muitos ângulos mortos. Alem disso, estes sistemas são também por norma sistemas caros.

Após descritos os problemas existentes pretende-se que o utilizador usufrua das seguintes funcionalidades do sistema:

- Deteção de situação de fogo através de técnicas de *machine learning*;
- Streaming da classificação atribuída pelo classificador;
- Alertas através de notificações remotas;
- Sistema energeticamente autossuficiente;

1.3 Solução proposta para resolver o problema

A solução proposta para a resolução do problema mencionado anteriormente foi a captação de imagens através de uma câmara de baixo custo com movimento de 180 graus e aplicação de algoritmos de visão computacional. O sistema é alimentado por um módulo de energia renovável solar.

Foi montada uma câmara de ligada a um *RaspberryPi 3* com capacidades de *pan&tilt* que por sua vez está dotado de um portal web que permite a utilizadores com permissões aceder ao streaming das imagens captadas pelo sistema

Foi também desenvolvida uma aplicação para *Android* que permite a autenticação do utilizador no sistema, receção de notificações remotas através do serviço *Firebase* da *Google* do streaming. De modo a tornar o sistema autossustentável, foi criado um módulo que permite atribuir uma certa autossuficiência ao sistema através de painéis fotovoltaicos.

O objetivo deste projeto foi criar um sistema que fosse capaz de detetar fogo e fumo através de vídeo em tempo real e conseguir com sucesso avisar as autoridades competentes ou dono do sistema com o mínimo de atraso possível.

1.4 Contribuição do projeto para o estado da arte

A deteção de fogo e fumo é um tema muito vasto, tendo pela comunidade científica já vários estudos publicados e disponíveis sobre diferentes técnicas. Foram estudados métodos de deteção através de sistemas *LIDAR*(*Light detection and ranging*) sensores, processamento de imagem e algoritmos de *machine learning*.

A contribuição para o estado da arte da temática abordada vai no sentido da exploração de diferentes combinações para processamento de imagem e redes neurais convulsionais.

Também, o projeto, tem como objetivo, ser implementado em hardware *low-cost*, sendo constituído por um *RaspberryPi* 3 capacitado de uma câmara, e uns motores para a função de pan&tilt, alimentados por uma bateria que por sua vez é também ela alimentada por painéis solares.

Este sistema, contribuí para o estado da arte com um conjunto de algoritmos de deteção com elevada fiabilidade e certeza em equipamento de baixo custo.

1.5 Organização do relatório

O presente relatório encontra-se dividido em 7 capítulos. No primeiro capítulo é feita uma introdução ao trabalho desenvolvido, definido o problema, e proposta uma solução para a sua resolução.

O segundo capítulo, é destinado ao estudo da técnica do projeto, mencionando com detalhe os projetos estudados ao longo do trabalho que envolvam tecnologias e/ou objetivos semelhantes. É também no segundo capítulo, feita uma comparação entre diferentes técnicas.

O terceiro capítulo é dedicado à exposição do estudo efetuado durante a realização do trabalho exposto no presente relatório. Neste capítulo são explicados os dois temas de maior relevância neste trabalho. O processamento de imagem, técnica que permite extrair indicadores de uma imagem, e a aprendizagem máquina que permite atribuir a um programa a habilidade de aprender.

O quarto capítulo apresenta a análise de requisitos feita do sistema. Neste capítulo são detalhados os requisitos técnicos ao funcionamento do sistema.

No quinto capítulo é exposto ao pormenor o sistema desenvolvido. Aqui, é apresentada a arquitetura da solução final e descrição dos módulos do sistema.

O sexto capítulo é destinado a testes e resultados. Depois de serem cumpridos os objetivos do trabalho, o mesmo foi exposto a um conjunto de testes de modo a comparar resultados com resultados já existentes e poder dar uma contribuição para a comunidade científica, conseguido desta forma afirmar o método aqui exposto na realização do trabalho.

No sétimo e último capítulo é feito um resumo de conclusão do trabalho assim como apresentados futuros melhoramentos do sistema.

2 Trabalhos relacionados

No presente capítulo é descrito o objetivo da visão computacional e trabalhos relacionados com objetivos semelhantes. Foram estudados sistemas desenvolvidos com fins académicos ou como prova de conceito (Cap. 2.1) e sistemas comerciais existentes e com resultados

comprovados (Cap. 2.2). As imagens apresentadas neste capítulo são todas retiradas da bibliografia de cada secção.

2.1 Exemplos de trabalhos semelhantes

A metodologia para escolher o conjunto de sistemas e algoritmos utilizados no projeto, foi um estudo de técnica.

Foram estudados sistemas comerciais existentes e estudos sobre o tema. Os artigos que foram estudados dividem-se em várias categorias que serão explicadas no próximo subcapítulo:

- Métodos baseados em *Bayesian networks*;
- Métodos baseados em máquinas vetoriais;
- Métodos baseados em redes neurais;
- Métodos baseados em sistemas *LIDAR*;
- Métodos baseados em sensores;

Após o estudo efetuado, ficou perceptível de que existem vários meios para atingir o mesmo fim. Da mesma forma, todos têm as suas particularidades e desvantagens.

2.1.1 Métodos baseados em Bayesian Networks

O sistema proposto utiliza redes probabilísticas ou *Bayesian Network* [6]s. A verificação é feita em vários passos como demonstra a imagem seguinte.

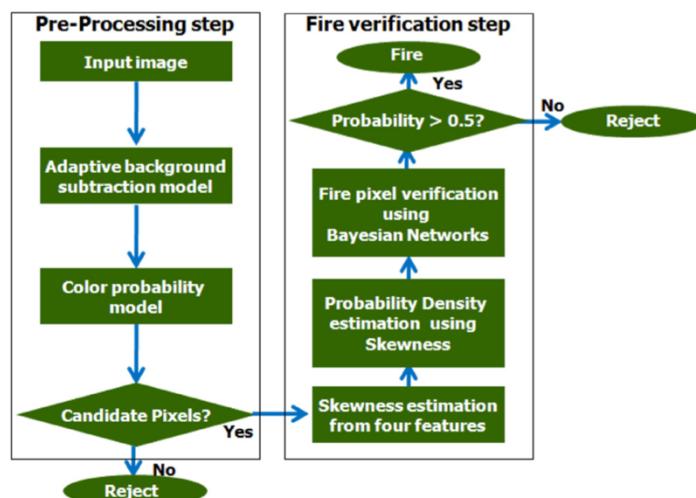


FIGURA 1 - BAYESIAN NETWORKS.

Em primeiro lugar é separado o *background* do *foreground*. Com esta técnica o autor do artigo consegue separar os pixéis candidatos a pixéis de fogo, dos pixéis que não são de fogo. Esta técnica é possível porque é sabido que o fogo está em constante alteração de forma, e assim é possível construir um algoritmo para identificar tudo o que altera a forma como sendo de *foreground*.

A segunda técnica, prende-se com a deteção de pixéis com as mesmas cores da gama do fogo. Mesmo após a separação do *background* e do *foreground*, e por não ser linear que seja só o fogo detetado como *foreground*, são também separados pixéis que não pertencem a essa categoria. Para conseguir filtrar os pixéis considerados barulhos na imagem, o autor utiliza um mapa de cores em probabilidades unimodais de *Gauss*. Essas probabilidades são calculadas através de um *dataset* de imagens que contêm fogo. O pixel é depois comparado com a probabilidade e o algoritmo capaz de decidir se este é ou não fogo.

Após a ultima verificação, o autor utiliza um módulo de redes probabilísticas. Apesar da filtragem baseada em processamento de imagem, o algoritmo, segundo o autor do artigo, não era capaz de distinguir objetos da cor vermelha em movimento, de fogo. Foram feitos testes com vários filmes em ambientes diferentes como mostra a imagem seguinte.

Video sequence	Number of frames	Description
Movie 1	491	Burning truck (outdoor)
Movie 2	750	Fire in garden (outdoor)
Movie 3	324	Burning bed (indoor)
Movie 4	241	Burning Christmas tree (indoor)
Movie 5	222	Burning tree (outdoor)
Movie 6	549	Laptop battery fire (indoor)
Movie 7	329	Burning foam mattress (indoor)
Movie 8	120	Fire-colored moving truck (outdoor)
Movie 9	165	Three men walking outside (outdoor)
Movie 10	412	Three men walking in a hallway (indoor)
Movie 11	393	Crowded parking lot (outdoor)
Movie 12	182	Accident in tunnel (indoor)

FIGURA 2 - BAYESIAN NETWORKS, DATASET.

E comparados os verdadeiros positivos e falsos positivos com dois métodos descritos no artigo (*Toeyin* e de *Ko*) como mostram as duas imagens seguintes.

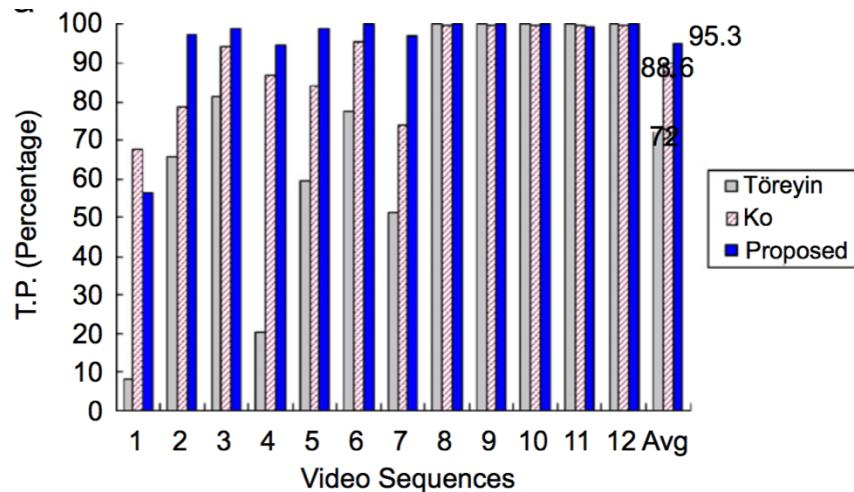


FIGURA 3 - BAYESIAN NETWORKS, COMPARAÇÃO COM OUTROS MÉTODOS (VERDADEIROS POSITIVOS).

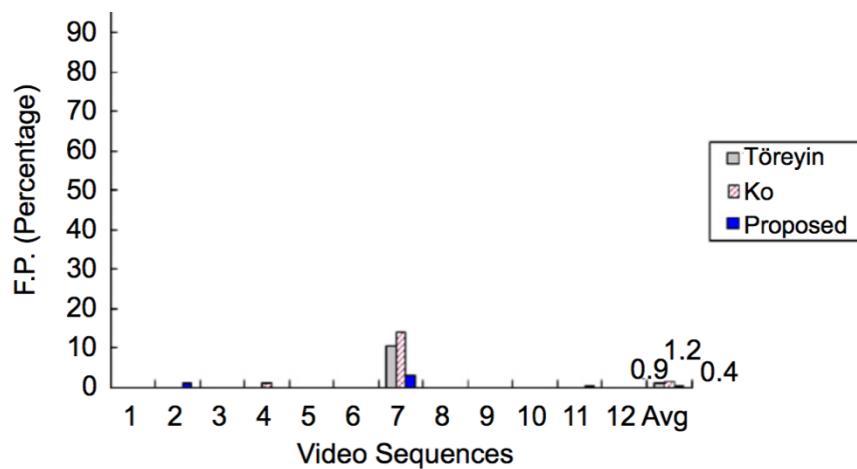


FIGURA 4 - BAYESIAN NETWORKS, COMPARAÇÃO COM OUTROS MÉTODOS (FALSOS POSITIVOS).

Por ultimo, foi feita a comparação em termos de rapidez, utilizando como medida os *FPS* (*Frames per secound*) em cada um dos 12 filmes. Como mostra a imagem seguinte.

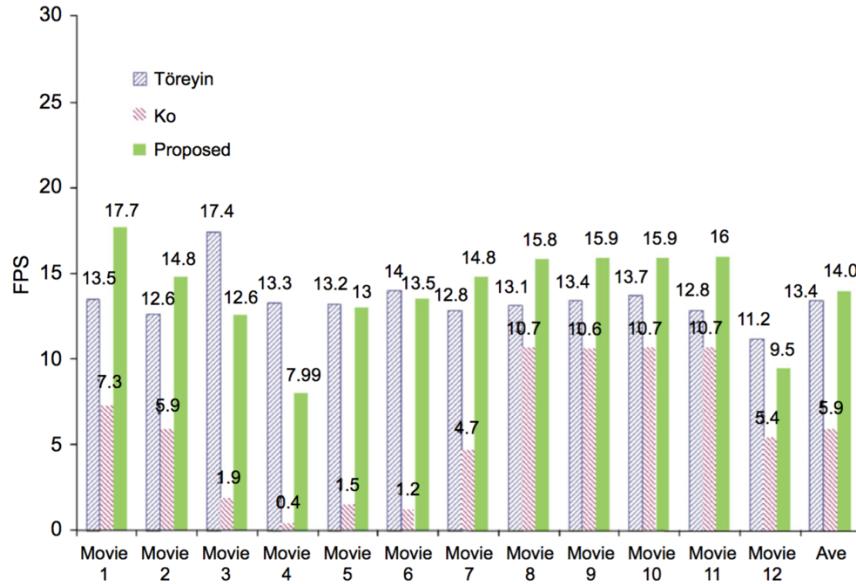


FIGURA 5 - BAYESIAN NETWORKS, COMPARAÇÃO COM OUTROS MÉTODOS (RAPIDEZ).

O anterior gráfico, contem alem dos 12 filmes de *dataset*, uma ultima coluna que mostra em média qual a quantidade de *FPS* (Frames per secound) que o algoritmo conseguiu processar.

2.1.2 Método baseados em máquinas vetoriais

O sistema proposto propõe um sistema baseado em máquinas vetoriais [10]. O sistema propõe resolver o problema por câmaras de circuito fechado através da deteção pelas câmaras e posterior envio por *SMS* (*Short message servisse*) e *streaming*. Na seguinte imagem podemos ver um diagrama da arquitetura geral da solução.

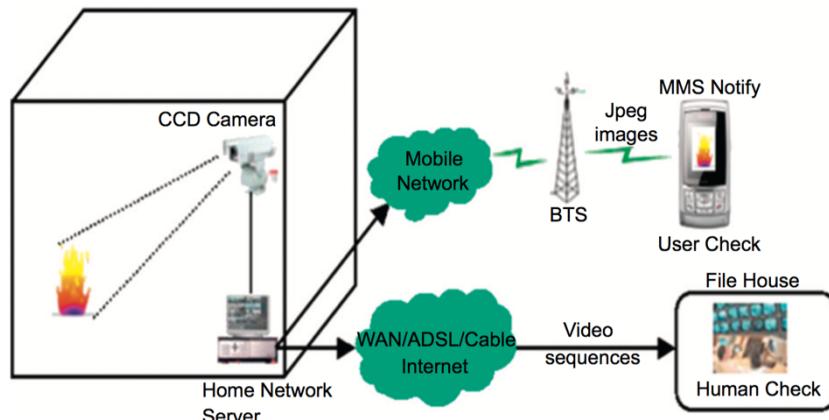


FIGURA 6 - SVM, ARQUITETURA DA SOLUÇÃO.

A verificação é feita em vários passos como se pode ver na imagem seguinte.

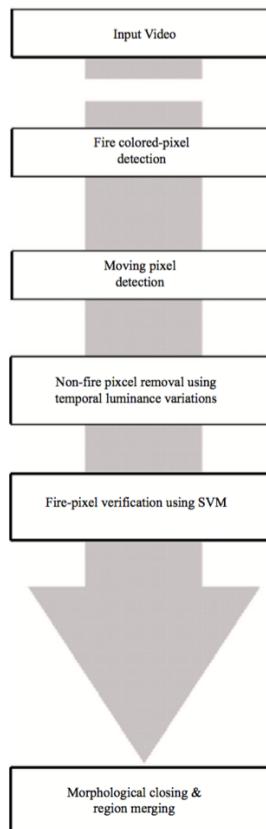


FIGURA 7 - SVM, ALGORITMO.

O primeiro passo é a tradicional deteção por pixéis de cores candidatas a pixéis de fogo. Esta deteção é feita com o pressuposto que uma região de fogo é mais clara que os vizinhos tanto em outdoor como em indoor. Para detetar os pixéis considerados como sendo fogo na imagem, o autor utiliza um mapa de cores em probabilidades uni modais de *Gauss*. Essas probabilidades são calculadas através de um *dataset* de imagens que contêm fogo. O pixel é depois comparado com a probabilidade e o algoritmo capaz de decidir se este é ou não fogo.

De seguida, é detetado o movimento com o objetivo de filtrar pixéis que não são fogo fazendo a diferença entre as localizações do pixel em dois *frames*. Se a diferença exceder um determinado *threshold* é considerado fogo, caso contrário, não é.

Apesar do sucesso na utilização dos dois métodos anteriores, é difícil para o algoritmo detetar objetos não fogo, mas com a mesma cor, e que se movem. Para suprimir a necessidade de ter um algoritmo que seja capaz de o fazer, foi utilizado um mapa de variações de luminosidade baseando-se no pressuposto que o fogo tem mais luminosidade. Para gerar o mapa, são utilizados 10 *frames* como *dataset*.

O ultimo passo, consiste na verificação por uma maquina vetorial ou *SVM* (Support vector machine). Foram utilizados 12 vídeos como *dataset* para validação do sistema. Os vídeos estão descritos na seguinte imagem.

Video sequence	Number of frames	Description
Movie 1	510	Man in fire-colored shirt behind fire
Movie 2	368	Burning tree
Movie 3	491	Burning truck
Movie 4	213	Fire in garden
Movie 5	190	Fire in forest
Movie 6	324	Burning bed
Movie 7	241	Burning Christmas tree
Movie 8	120	Fire-colored moving truck
Movie 9	165	Three men walking on ground
Movie 10	412	Three men walking in hallway
Movie 11	393	Accident in tunnel
Movie 12	394	Dancing man in fire-colored shirt

FIGURA 8 - SVM, DATASET.

Este sistema é comparado com o método de *Toreyin*, o qual teve os resultados demonstrados na seguinte figura utilizando os mesmos vídeos.

	Detection rate	True positive	False positive	Missing rate
Movie 1	34.0	34.0	0.0	66.0
Movie 2	87.5	82.6	4.9	7.6
Movie 3	7.9	7.9	0.0	92.1
Movie 4	51.2	7.1	44.1	4.7
Movie 5	73.7	63.7	10.0	16.3
Movie 6	81.2	81.2	0.0	18.8
Movie 7	20.3	20.3	0.0	79.7
Movie 8	100	100	0.0	0.0
Movie 9	100	100	0.0	0.0
Movie 10	100	100	0.0	0.0
Movie 11	100	100	0.0	0.0
Movie 12	100	100	0.0	0.0
Average	71.3	66.4	4.9	23.7

FIGURA 9 - SVM, RESULTADOS DO MÉTODO DE TOREYIN.

Os resultados conseguidos pelo sistema desenvolvido, mostraram-se bastante superiores como demonstrado na seguinte imagem.

	Detection rate	True positive	False positive	Missing rate
Movie 1	57.2	55.2	2.0	40.8
Movie 2	77.7	77.7	0.0	22.3
Movie 3	67.8	67.8	0.0	32.2
Movie 4	56.3	56.3	0.0	43.7
Movie 5	97.9	97.9	0.0	2.1
Movie 6	94.4	94.4	0.0	5.6
Movie 7	87.6	86.8	0.8	11.6
Movie 8	100	100	0.0	0.0
Movie 9	100	100	0.0	0.0
Movie 10	100	100	0.0	0.0
Movie 11	98.5	97.0	1.5	0.0
Movie 12	100	100	0.0	0.0
Average	86.5	86.1	0.4	13.2

FIGURA 10 - SVM, COMPARAÇÃO COM O MÉTODO DE TOREYIN.

2.1.3 Métodos baseados em redes neurais

O sistema proposto propõe um sistema baseado em redes neurais [9] para resolver o problema da deteção de fogo. O método combina vários métodos de processamento de imagem para deteção de fogo e de fumo e uma rede neuronal *feedforward*.

Primeiro, é feita uma extração baseada na cor do fogo para eliminar interferências baseadas na cor. Depois são calculados os vetores da imagem com base na forma e na cor. De seguida, é utilizada uma rede neuronal para deteção da chama.

O autor faz o processamento da imagem baseando-se em dois conceitos. Características estáticas e dinâmicas da imagem. Para a categoria de característica estática, o autor utiliza a cor, forma e textura. Neste caso isolado, a cor está associada à temperatura. Com o aumento da temperatura a cor varia de vermelho saturado para menos saturado.

É retirado da imagem o vetor da textura. Por ultimo, como característica estática é retirado o vetor da forma baseado em equações de Fourier. É ainda utilizado um vetor baseado na área da região, media e *centroid* da forma do fogo.

Como dinâmico é considerado o deslocamento da área no plano, a mudança de limites e a mudança de forma A mudança da área no plano supõe que a área do fogo muda muito no inicio do fogo. A área pode então ser calculada com o numero de pixéis com um valor superior a um *threshold*. Porem só este método pode levar a erro quando existem objetos não fogo que se movem. É também detetado o movimento geral do fogo através da análise de uma sequencia de imagens. É também utilizada a deteção da mudança dos limites. No inicio do fogo, os cantos das chamas mudam regularmente, ficando estável com o passar do tempo. O vetor da mudança dos limites na imagem vai ser retirado baseando-se na forma e curvatura do limite.

Existem dois processos. A segmentação da imagem que compreende a separação do objeto da imagem e identificação das partes necessárias para serem processadas. O autor utiliza uma segmentação baseada em *thresholds*. O segundo processo é o reconhecimento da imagem. Este processo pela análise, deteção, e reconhecimento da imagem contextualizando-a. A segmentação tem dois processos filho. A extração baseada em cor por o fogo ter uma cor bastante característica. Primeiro retira os valores *raw* do pixel da imagem. São depois convertidos para o espaço de cores RGB e analisado cada canal com base na equação da seguinte imagem.

$$V(R, G, B) = \begin{cases} 117 \leq R \leq 255 \\ 88 \leq G \leq 255 \\ 44 \leq B \leq 255 \end{cases}$$

FIGURA 11 - REDES NEURONAIAS, SEGMENTAÇÃO POR COR.

A segmentação do fogo da imagem utiliza a segmentação por regiões da imagem e métodos para segmentar o fogo em si.

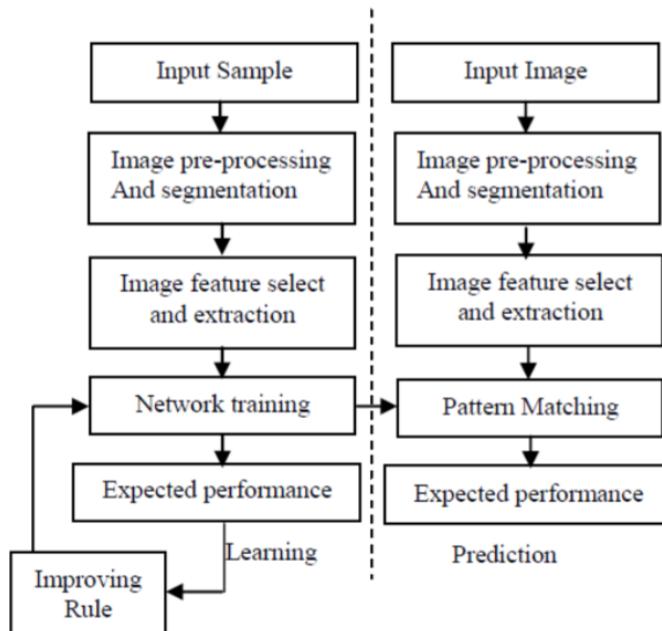
A imagem é dividida em 25 regiões que são analisadas para ver a sua categoria. São depois juntadas as várias regiões da mesma categoria. De seguida o fogo é segmentado verificando todos os pixéis na mesma área até encontrar um pixel suspeito de ser fogo. Uma vez encontrado, são procurados pixéis vizinhos com intensidades semelhantes. O numero de pixéis encontrados é comparado a um *threshold*. Se for maior, pertence a área de fogo.

O ultimo passo é a decisão do reconhecimento sem interferência humana baseado numa rede neuronal. O autor, utiliza como input da rede neuronal os *feature vectores* extraídos nas fases anteriores de convergidos (cor e forma). Apesar de não existir um método formal para determinar a camada escondida (ou *hidden layer*) o autor apoia-se na equação da seguinte para encontrar o balanço entre o alto e baixo numero de neurônios na camada escondida.

$$y = \frac{\frac{1}{2}c \times r^2 + \frac{3}{2}c \times r - 1}{c + r}$$

FIGURA 12 - REDES NEURONAIAS, ESCOLHA DO TAMANHO DA CAMADA OCULTA.

A equação tem em conta o facto de quanto mais neurônios tiver a camada oculta da rede neuronal, maior o poder de processamento exigido ao hardware, maior o risco de *over-learning* e maior a facilidade de convergência do algoritmo de aprendizagem *back propagation*. A camada de output tem três neurônios. O vetor {1,0,0} de output significa uma cena de fogo, o vetor {0,1,0} significa que a imagem tem objetos de interferência. O vetor {0,0,1} simboliza a presença de floresta. A figure seguinte mostra o algoritmo global do sistema

**FIGURA 13 - REDES NEURONAIAS, ALGORITMO.**

A rede neuronal, utiliza como algoritmo o *back propagation* baseado no ajuste de pesos e *bias* pelo *gradient descent*. A imagem seguinte mostra o treino da rede.

I_1	I_2	I_3	ρ	N	Y/N
.0457	.0245	.1894	.4277	.8326	N
.4279	.1356	.3274	.5282	.9433	Y
.2537	.3550	.0455	.3146	.6879	Y
.1343	.7467	.5465	.3476	.1447	Y
.0243	.7298	.4723	.4532	.2346	Y

FIGURA 14 - REDES NEURONAIAS, TREINO DA REDE.

As primeiras três colunas, representam os três canais de cores no espaço RGB, a quarta coluna representa a região suspeita de ser fogo. A quinta coluna representa o numero de cantos da fama. A próxima imagem ilustra o resultado dos testes.

I_1	I_2	I_3	ρ	N	Y/N	results
.6424	.0245	.1894	.4277	.8326	Y	.9946
.4279	.1356	.3274	.5282	.9433	Y	.9137
.5257	.1750	.0455	.3146	.7479	Y	.6763
.1356	.7467	.5468	.3466	.1467	Y	.9432
.0245	.4298	.6723	.4532	.2376	N	.7698

FIGURA 15 - REDES NEURONAIAS, RESULTADOS DOS TESTES.

O autor escolheu após o treino, um *threshold* de 0.8. Um resultado menor que 0.8 indica não fogo, maior indica fogo. O autor reporta um *recall* de 75%.

2.2 Sistemas comerciais

Neste subcapítulo são apresentados alguns sistemas comerciais encontrados no decorrer da investigação do projeto.

2.2.1 Firewatch

Firewatch [5] é um sistema terrestre de vigilância remota de incêndios na floresta que emite alarmes quando o fogo é detetado. Este sistema está atualmente em utilização na Alemanha em vários estados e em várias cidades da Europa. O sistema conta com as seguintes características:

- Deteção através de uma câmara montada numa torre;
- Deteção de nuvens de fumo de dia e de noite;
- Transmissão de dados via cabo ou rádio;
- Supervisão rápida de uma área de 70.000 hectares;
- Elevada qualidade de imagem;

A seguinte imagem mostra a arquitetura da solução FireWatch.

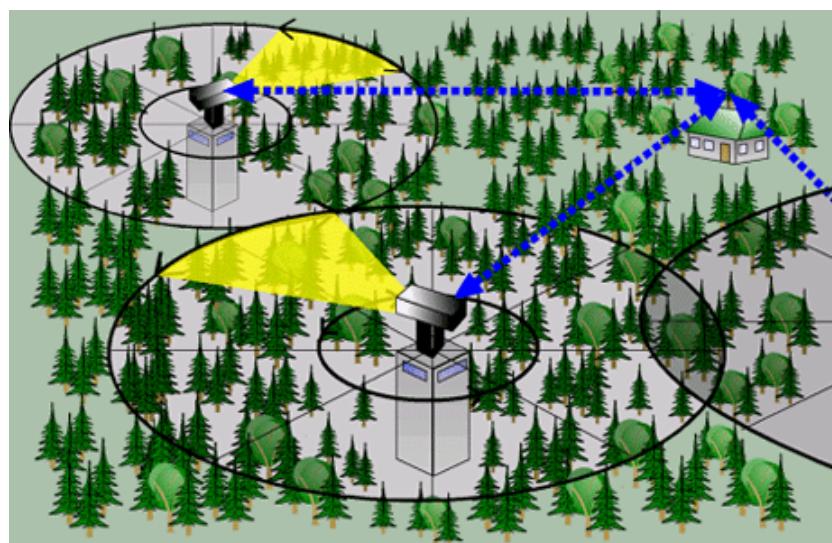


FIGURA 16 - FIREWATCH SENSORES.

Cada sensor ótico, é capaz de rodar 360º num espaço de 4 a 10 minutos com cada rotação a representar um ângulo de 10º. A cada posição, recolhe 3 imagens. Este sistema assenta na supervisão humana, não se preocupando com a taxa de erro do sistema.

2.2.2 ZeroFires

O ZeroFires [4] contém três soluções de combates a incêndios, sendo que uma se baseia em software e duas em hardware.

A solução *ZF-WildFire station series* é uma solução de deteção de fogos florestais desenhada para ser implementada em torres de vigia de múltiplas maneiras, incluindo em postes. A solução implementa um software com capacidade de analisar a imagem e procurar na mesma fogo ou fumo. Uma vez detetado um dos elementos é enviado um alarme (Web, e-mail ou SMS) por dados móveis, ou pelo sistema integrado de comunicações por satélite. O sistema é capaz de

detetar fogo de 1 metro quadrado e cada câmara consegue cobrir 12.500 km quadrados. Em relação ao sistema anterior este é claramente menos abrangente. Mas em questão de rapidez, este é capaz de processar 2000 km quadrados por minuto. Estas estações são alimentadas por painéis solares e energia solar. A seguinte imagem, mostra a instalação deste sistema.



FIGURA 17 - ZF-WILDFIRE STATION SERIES.

A segunda solução chamada **ZF-Portable system** é semelhante à anterior solução, mas esta é portátil. A solução foi desenvolvida para poder ser facilmente transportável. Em termos de características técnicas, é capaz de detetar fogos de 1 metro quadrado num espaço de 18 km quadrados ou fogos de 12 metros quadrados num espaço de 12 km quadrados. A seguinte imagem, mostra a solução descrita.



FIGURA 18 - ZF-PORTABLE SYSTEM.

A terceira solução chamada de **ZF-Dashboard** é uma solução baseada na comunicação e interface para o utilizador com os sistemas anteriormente descritos. Este software permite a comunicação com as câmaras e definição de parâmetros como alarmes, velocidade da câmara entre outros. Este software está adaptado para dispositivos móveis e desktop. A próxima imagem ilustra a aplicação.



FIGURA 19 - ZF-DASHBOARD.

3 Tecnologias e áreas estudadas

Neste capítulo, são descritas as tecnologias estudadas no desenvolvimento do projeto. Todo o conhecimento adquirido no âmbito deste projeto foi com base num livro *open-source* [1] de redes neurais, num curso de visão computacional e *machine learning* [2] e através de um curso de carros autónomos [3]. Também as imagens destes capítulos são da autoria do livro e

cursos mencionados. Como já referido, o trabalho descrito no presente relatório assenta sobre o tema visão computacional. Esse tema é dividido em duas grandes áreas. O processamento de imagem, e a aprendizagem máquina.

3.1 Processamento de imagem

O processamento de imagem é o método de conseguir através de operações sobre a mesma, extrair desta informação útil. Neste tipo de técnica, por norma, o input é uma imagem, e o output são as características extraídas.

O processamento de imagem inclui tipicamente os três passos seguintes:

- Aquisição da imagem;
- Análise e manipulação da imagem;
- Output do processamento;

O problema que o processamento de imagem tenta resolver é o representar de uma imagem e extração do mesmo tipo de informação que o ser humano consegue ter retirar. Para o ser humano é trivial distinguir por exemplo que a imagem seguinte tem dois logotipos e um corresponde ao logotipo da biblioteca *openCv* e o outro da linguagem de programação *python*.



FIGURA 20 - LOGOTIPO OPENCV E PYTHON.

O computador, ao analisar uma imagem, não tem percepção do que representam os conteúdos visuais. A câmara capta a reflexão da luz sobre determinado objeto e representa a imagem sobre a forma de três matrizes de inteiros de 8 bits por cada posição. Essas três matrizes representam um canal de um esquema de cor. Por exemplo, caso seja utilizado o espaço de cores *RGB*(RedGreenBlue) cada uma das matrizes representa respetivamente o *red*, *green* e *blue*. Existem, no entanto, outros que tentam imitar melhor a cor e ser mais intuitivos como o exemplo o *HSV*(*HueSaturationValue*) entre outros.

Cada pixel representa um valor de 0 a 255, onde 0 é a ausência de cor e 255 é o branco. Por exemplo, com o vetor numa posição (x,y):

- O vetor [0,0,0] seria um preto;
- O vetor [255,255,255] seria um branco;
- O vetor [255,0,0] seria um vermelho;
- O vetor [0,255,0] seria um verde;
- O vetor [0,0,255] seria um azul;

A imagem seguinte mostra como é vista uma imagem de 5*5 pixéis do ponto de vista do computador em que cada matriz representa um dos três canais *RGB*.

54	58	255	8	0
45	0	78	51	100
85	47	34	185	207
22	20	148	52	24
52	36	250	74	214
158	0	78	51	247
	72	74	136	255
				74

FIGURA 21 - IMAGEM DO PONTO DO VISTA DO COMPUTADOR.

3.1.1 OpenCv

O processamento de imagem, como referido no ponto 3.1, compreende operações entre matrizes para a extração de indicadores. Essas operações são extremamente intensivas em termos matemáticos e como tal, é muito fácil escalar ao ponto de não se conseguir obter os melhores resultados e ter um compromisso elevado com a performance.

Para colmatar esse entrave, entre várias, foi desenvolvida a biblioteca *OpenCv*. Esta Framework contem de uma forma muito organizada os principais métodos de processamento de imagem onde se destacam os seguintes:

- Operações básicas como a leitura, escrita e visualização de imagens;
- Conversão entre espaços de cor;
- Acesso fácil a todas as características da imagem como cor de um pixel, regiões, forma entre outros;
- *Gradients, kernels, smoothing e blurring, deteção de limites, contours e histogramas;*
- *Canny edges;*
- Redimensionamento, *thresholding*, escala entre outros;

- Operações aritméticas e morfológicas entre imagens;
- *Image Descriptors*;

Existem outros métodos que se podem ver na documentação da biblioteca. Também para ajudar nas operações algébricas, é muitas vezes utilizada a biblioteca *numpy*. No âmbito deste projeto, ambas as bibliotecas são utilizadas.

3.1.2 Aprendizagem máquina

O problema dos fogos, representa um problema de classificação. O principal requisito foi que o sistema fosse capaz de distinguir entre um cenário com fogo e um cenário sem fogo. Entre outras técnicas, são utilizados para a resolução deste tipo de problemas métodos de *machine learning* ou aprendizagem máquina. Esta técnica permite através da implementação de algoritmos fazer o computador aprender sobre um tema sem ser explicitamente programado para o fazer. Existem dois tipos de aprendizagem. *Supervised learning* ou aprendizagem supervisionada, e *unsupervised learning* ou aprendizagem não supervisionada.

A **aprendizagem supervisionada**, implica o conhecimento prévio dos elementos utilizados para o treino do sistema através de *labels* ou categorias de cada um dos elementos. Num exemplo prático, se quisermos reconhecer na imagem uma cadeira ou uma mesa, esses dados seriam dados como input ao algoritmo juntamente com a respetiva descrição. Dessa forma, o algoritmo aprende com conhecimento prévio o que significa cada imagem. As imagens passadas ou qualquer outro tipo de dado para aprendizagem é chamado de dataset. O conceito de *dataset* é explicado nos próximos parágrafos.

A **aprendizagem não supervisionada** é utilizada para agrupar em categoria elementos não conhecidos. Sintetizando e como exemplo damos um *dataset* sem descrição em que existem duas categorias de conteúdos fotográficos que nós sabemos. Cem fotografias de lagos, e duzentas fotografias de campos agrícolas. Neste cenário, é possível desenvolver um algoritmo que sem qualquer conhecimento consiga classificar e agrupar em duas categorias cada uma das imagens.

Antes da implementação do algoritmo é necessário reunir um *dataset* com exemplos que queremos que o programa aprenda. Este *dataset* deverá ser grande e o mais diversificado possível para aumentar a precisão do algoritmo. Após reunido o *dataset* é necessário proceder à sua divisão para passar até três fases deste tipo de sistemas.

A primeira fase é a fase de treino. Nesta fase o utilizador de todo o *dataset*, seleciona uma percentagem que vai utilizar para o algoritmo de *machine learning* aprender.

A segunda fase, e esta é uma fase opcional, é a fase de validação. Esta fase é utilizada dependendo do algoritmo a utilizar. Os algoritmos contam com parâmetros que alteram a sua performance. Estes parâmetros são chamados de *hyperparameters*. Estes *hyperparameters* são por vezes um pouco aleatórios e só são conseguidos através de tentativa erro. Para selecionar os melhores *hyperparameters* é utilizada a fase de validação. Esta fase utiliza uma pequena percentagem do *dataset* de treino (tipicamente 5 ou 10%) para conseguir testar os vários *hyperparameters* e escolher os melhores.

A ultima fase, é a fase em que é validada a aprendizagem. Esta fase chama-se de fase de teste. Nesta fase é selecionada uma percentagem não maior que 33% do *dataset* que é utilizada o algoritmo prever os valores de input consoante a aprendizagem feita na primeira fase. Na seguinte imagem podemos ver valores típicos utilizados para a divisão do dataset.

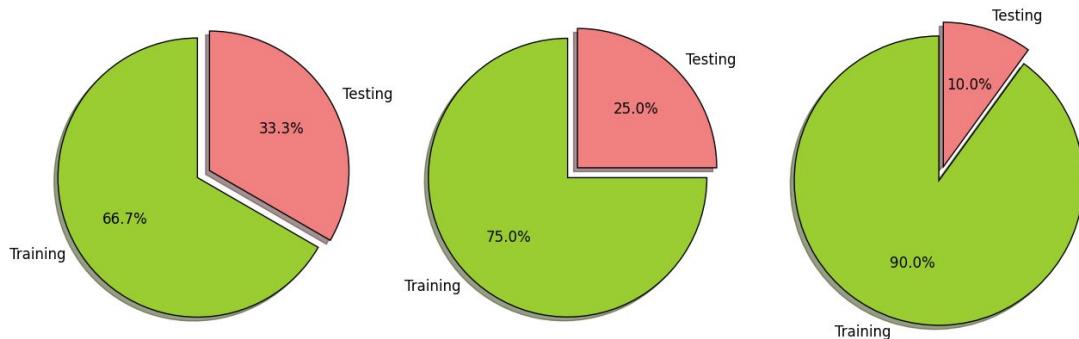


FIGURA 22 - DIVISÃO DO DATASET.

Em qualquer um dos casos, é extremamente importante que o *dataset* ao ser dividido seja isolando, evitando por exemplo elementos de treino na fase de teste. Caso isso acontecesse, estaríamos a treinar o algoritmo para passar na fase de teste.

A aprendizagem máquina é uma área muito vasta que implica muito estudo para ser dominada. Existem atualmente vários algoritmos que poderiam com diferenças de performance solucionar o problema no presente relatório exposto.

3.1.3 Support Vector Machines

SVM's ou *support vector machines* é um dos temas em *machine learning* sobre o qual se pode escrever por horas e ter muito a dizer. O algoritmo é extensivamente matemático, e é fácil nos pertermos no estudo do mesmo.

A razão das *SVM's* serem tão populares, é que é possível ter um bom classificador sem pertermos muito tempo a afinar os *hyperparameters* ou parâmetros da rede.

Para entender como funcionam as *SVM's* é necessário entender primeiro o conceito de separação linear. A separação linear acontece quando, num par com dois conjuntos, é possível traçar uma reta e separar os dois como ilustra a seguinte imagem.

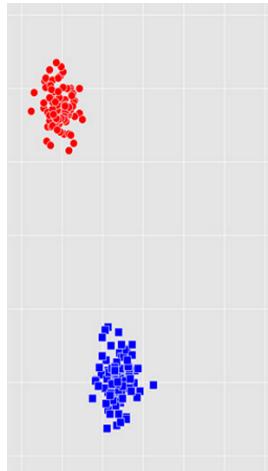


FIGURA 23 - SEPARAÇÃO LINEAR POSSÍVEL.

Neste caso podemos claramente separar as duas classes com uma reta. Um exemplo de classes que não são linearmente separáveis é ilustrado na seguinte figura.

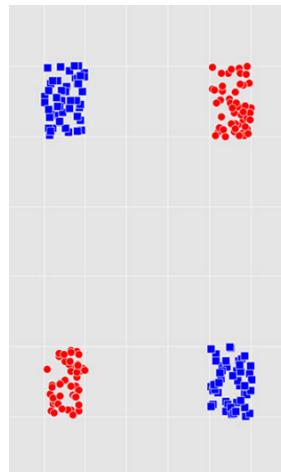


FIGURA 24 - SEPARAÇÃO LINEAR IMPOSSÍVEL.

É impossível, com uma só reta separar estas quatro classes. Podíamos eventualmente traçar uma segunda reta. Reta essa que iria sobrepor a primeira, o que também torna as classes linearmente não separáveis. Este é conhecido como o problema do *OR* exclusivo.

A linha utilizada para separar as classes, em duas dimensões é somente uma linha, em três, é um plano, e em mais que três dimensões é um *hyperplane*. Esta é a linha fronteira utilizada para classificar as classes. Quanto mais afastada uma classe estiver da linha, maior a confiança do algoritmo na classificação. A razão pela qual isto acontece é porque, existe uma maior margem de confiança o que dá uma maior flexibilidade quando classificamos os dados para fase de teste como demonstra a figura seguinte.

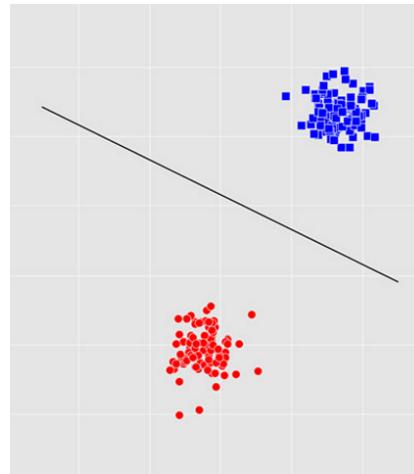


FIGURA 25 - LINHA FRONTEIRA.

A linha de fronteira é decidida com base nos *support vectors*, daí vem o nome do algoritmo. Os *support vectors* são os vetores de cada classe que se encontram o mais próximo da linha fronteira como mostram os elementos da próxima imagem.

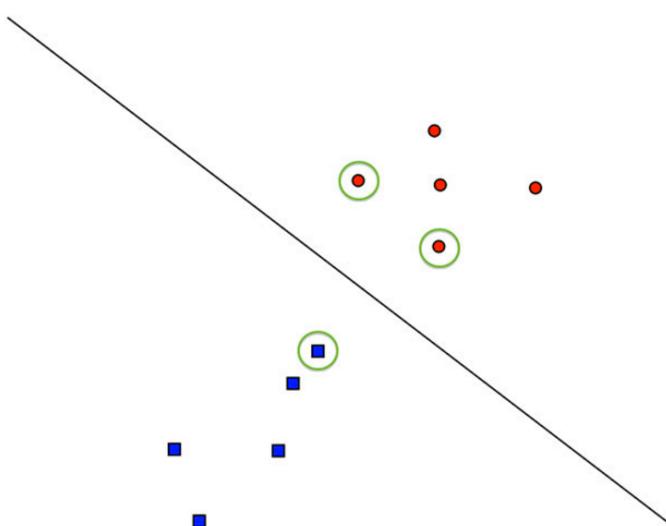


FIGURA 26 - SUPPORT VECTORS.

Com esses *support vectors*, é maximizada a margem da linha fronteira separando as classes de maneira ótima.

Uma parte do problema está endereçada. Como conseguir duas classes serem classificadas com boa confiança. A outra separar linearmente várias classes. Sem esse problema resolvido, o algoritmo não passaria de um classificador linear.

A ideia para a separação nesses casos é. Uma vez que não são separáveis no plano de duas dimensões, são projetados em mais de duas dimensões e calculado um *kernel*. Este *kernel*, é o *kernel* que vai otimizar a separação quando existem mais de duas classes.

O *kernel* a utilizar é muito importante no sentido que, cada *kernel* tem as suas especificidades. Por exemplo. O *kernel* linear é representado pela seguinte função:

Podemos ver que a função só tem como *hyperparameter* o T . Vendo a função polinomial. Vemos que a função polinomial tem mais dois *hyperparameters* para afinar.

Concluindo. O *SVM* é um algoritmo de *machine learning* muito poderoso, difícil de entender, mas fácil de utilizar. Dependendo do *kernel* escolhido, a fase de validação do algoritmo pode ser mais ou menos demorada.

Este é um algoritmo que é muito rápido a treinar mas lento a validar.

1.1.2. Decision trees e Random Forests

As *Decision trees* e *Random Forests* ambas técnicas de *machine learning* diferentes, mas que estão de alguma forma interligadas.

Às *decision trees* ou árvores de decisões é dado uma imagem de treino, extraído um *feature vector*, e criados nós de decisão. Ao contrário de outras técnicas, esta tem a vantagem de permitir a cada nível da árvore verificar o comportamento com a imagem facilitando por vezes o trabalho do programador. A seguinte figura mostra um exemplo de uma árvore de decisão.

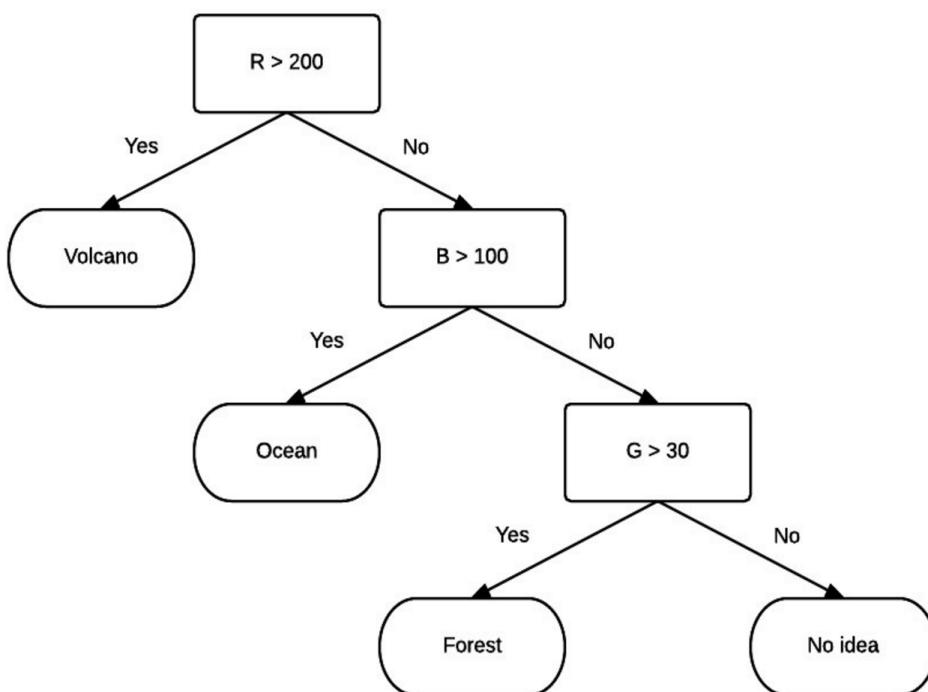


FIGURA 27 - ARVORE DE DECISÃO RGB.

No exemplo da imagem anterior, a árvore de decisão foi construída com base na média dos valores *RGB* da imagem de input. Como exemplo, uma imagem que extraísse o seguinte *feature vector*.

Esta imagem seria classificada de acordo com a árvore como sendo uma floresta. Estes *feature vectors* podem ser extraídos com vários algoritmos que constroem automaticamente a árvore através dos dados de treino.

O algoritmo utiliza o que se chama de *informative split* para depois de ter os *feature vectors* certos conseguir a melhor divisão da arvore tentando todas as *features* e vendo qual funciona melhor.

As *Random Forests* começam onde as arvores de decisão acabam. Alguém, após fazer várias arvores de decisão, decidiu construir uma arvore de arvores de decisão. As *Random Forests* encaixam na categoria de *ensemble methods*. Em vez de utilizarem um só classificador, utilizam muitos como mostra a imagem seguinte.

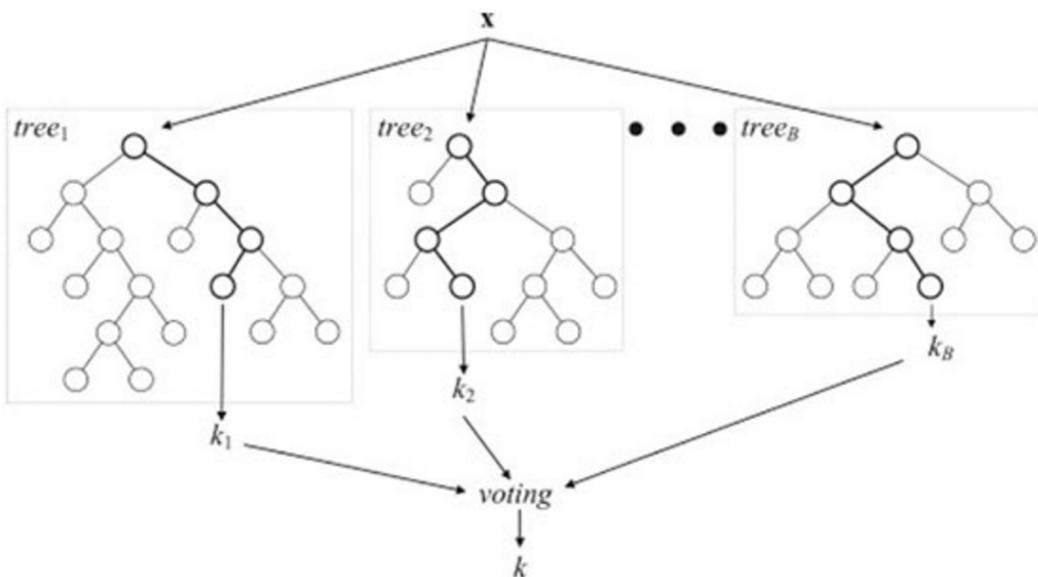


FIGURA 28 - RANDOM FOREST.

Cada arvore de decisão, vota para uma decisão final. A categoria com mais votos, ganha. O método alem de agregar vários resultados, adiciona também alguma aleatoriedade para o treino de cada arvore. Este passo é chamado de *bootstrapping*. De uma forma resumida, cada arvore de decisão vai contar com o numero de nós igual à metade das imagens de treino em que os *feature vectors* dessas imagens participam na construção da arvore. O **segundo passo** é atribuir mais um grau de aleatoriedade ao sistema. Supondo que temos um *feature vector* com dimensão de 128. Ao 128 aplicamos a raiz quadrada ou o logaritmo. O resultado dessa operação dita um novo *feature vector* para cada imagem. O passo final é o voto de cada arvore.

O algoritmo de *Random Forest*, é chamado em *machine learning* como um método de *ensemble* learning o que será descrito no próximo subcapítulo.

3.1.4 Aprendizagem Ensenble

O *ensemble learning* combina resultados de vários modelos para ter melhores resultados. O *Random Forest* é um exemplo de *ensemble learning* pois combina o resultado de várias arvores de decisões.

O *ensemble learning*, quando bem aplicada, supera em performance qualquer algoritmo de *machine learning*. Existem vários métodos de *ensemble learning* entre os quais os mais comuns:

- **Bagging** - Com um conjunto aleatório de exemplos dos dados de treino damos para diferentes versões do modelo. No exemplo do *random forest*, para diferentes árvores de decisão, e deixamos que votem na decisão final
- **Boosting** – Com esta técnica identificamos quais os pontos fracos do *dataset* e tenta torná-los fortes pegando nos pontos fracos identificados na fase de treino e passando-os a outras fases do algoritmo.
- **Bucket os models** – Este método tem em conta vários modelos sendo estes iguais ou diferentes e escolhe o melhor modelo
- **Stacking** – Igual ao modelo *Bucket of models*, mas em vez de escolher o melhor modelo, combina todos os resultados. Este é o exemplo do *random forest*.

3.1.5 Convolutional Neural Networks

Os métodos anteriores, apesar de serem métodos de *machine learning*, são em tudo diferentes das redes neuronais convulsionais no sentido em que os anteriores métodos se apoiam fortemente em matemática ao contrário deste.

Para falar em redes neuronais convulsionais temos de falar em **redes neuronais artificiais**. As redes neuronais artificiais foram desenvolvidas em analogia com a maior máquina de processamento conhecida. O cérebro! O cérebro contém vários neurónios que se interligam entre eles e transmitem impulsos elétricos. Esses neurónios “assimilam” conhecimento e estão constantemente em aprendizagem desde o momento em que nascemos até ao momento em que morremos. Para termos uma ideia do tamanho do cérebro, este é dividido em vários córtices. O córtex primário ou como é conhecido o V1, contém cerca de 140 milhões de neurónios para aprendizagem de conteúdo visual com biliões de ligações entre eles. Mais complexo fica, quando estudamos e constatamos que o córtex V2, V3, V4 e V5 também participam no reconhecimento da imagem.

As redes neuronais artificiais foram desenvolvidas com apoio na filosofia do cérebro sendo o elemento mais básico da rede o neurónio. O primeiro tipo de rede a ser desenvolvido, foi uma rede de *perceptrons* o que deu origem às *Multi-layer Perceptron Networks* ou *MLP's*. O *perceptron* na sua forma mais básica, tem vários inputs e pode fornecer um ou mais outputs. Para compreendermos o exemplo seguinte demonstra um *perceptron* com inputs x_1 e x_2 .

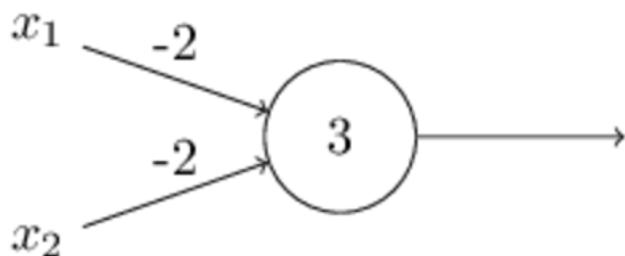


FIGURA 29 – PERCEPTRON.

Os inputs do *perceptron* são depois ponderados com um peso (-2) e enviados para o neurónio. O neurónio tem implícito um valor chamado Bias (3). Concluindo, o resultado de um *perceptron* é o somatório da multiplicação dos inputs com os pesos da ligação e soma do *bias* do neurónio.

O *bias* é simplesmente um valor que indica o quanto fácil é o neurónio disparar a 1 ou a 0. Depois de disparado, e caso seja a ultima camada ou camada de output da rede neuronal, o resultado passa por uma função de ativação que vai indicar o valor final da rede. A imagem seguinte mostra o funcionamento simples da rede neuronal do input ao output a mais baixo nível mostrando a função de ativação do *perceptron*.

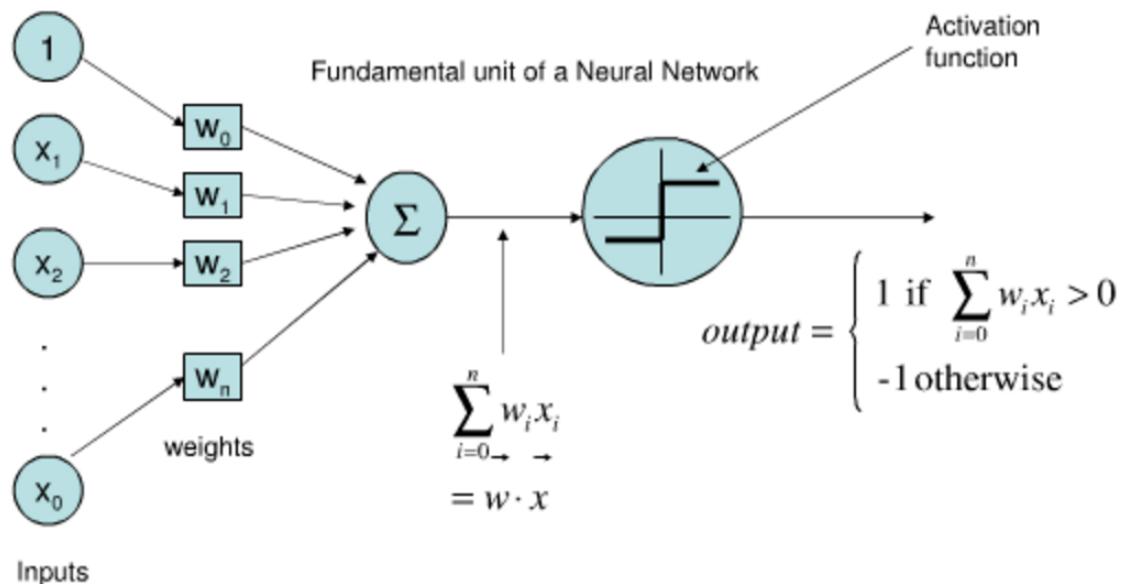


FIGURA 30 - ARQUITETURA DE REDE NEURONAL.

Esta função de ativação chama-se de função *Step function* ou função de passo que será explicada nos próximos parágrafos.

Existem ainda dois conceitos a serem explorados. O conceito de *layers* e o neurónio do tipo *sigmoid*.

A rede neuronal seria extremamente simples se contasse apenas com um neurónio e também extremamente ineficaz no que toca a problemas complexos. Na imagem 31 ilustramos uma porta *NAND* que é resolvida com um *perceptron*. Mas se quiséssemos por exemplo adicionar

dois bits com portas NAND, não conseguiríamos com aquele modelo. A seguinte imagem ilustra quais seriam os requisitos do problema.

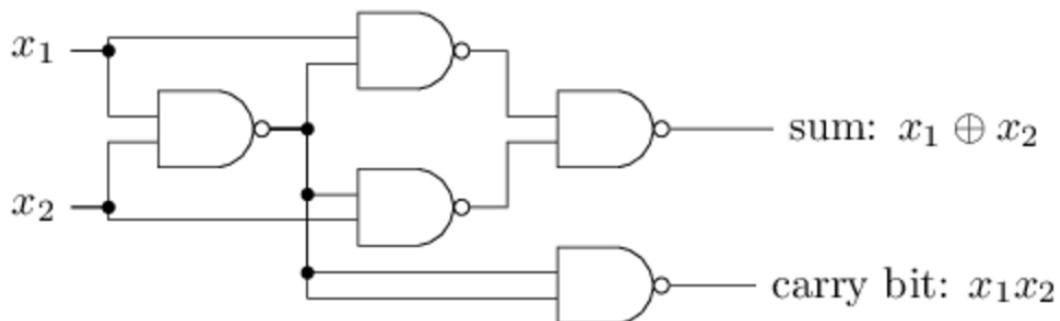


FIGURA 31 - SOMA DE DOIS BITS COM NAND'S - PONTO DE VISTA DIGITAL.

Cada *perceptron* só consegue neste caso modelar uma porta. Nesse sentido facilmente nos deparamos com um problema. Para o resolver, as redes neurais contam com não um, mas vários neurônios divididos em várias camadas.

- **A camada de input.** Esta é a primeira camada que recebe os inputs;
- **A camada oculta.** Esta camada é onde é feita toda a computação. Multiplicados os pesos e adicionados os *bias*. Podem existir várias camadas ocultas;
- **A camada de output.** Esta é a camada responsável por dar um resultado da rede;

A seguinte imagem mostra uma rede neuronal das várias camadas. Esta rede resolve o problema de adição de dois bits com portas *NAND*.

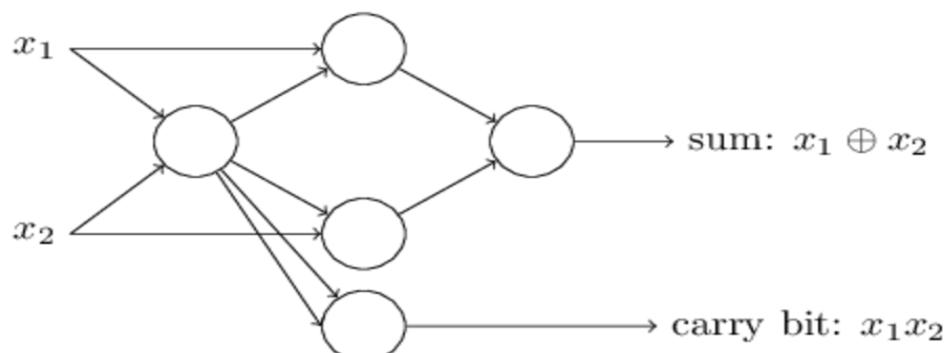


FIGURA 32 - SOMA DE DOIS BITS COM NAND'S - PONTO DE VISTA NEURONAL.

A próxima evolução, foi a introdução da função *sigmoid*. A função do *perceptron* é como representa a seguinte imagem.

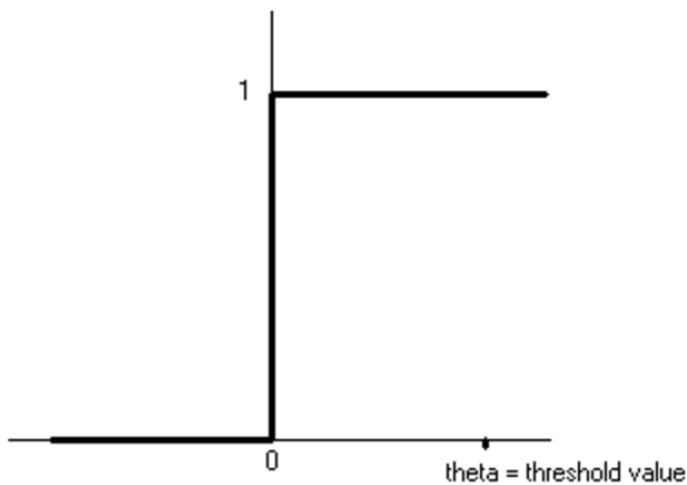


FIGURA 33 - FUNÇÃO DE PASSO.

Esta função introduz um problema na rede que é comum com os algoritmos de *machine learning*. A aprendizagem. Com esta função não é possível atribuir um grau suficientemente bom de aprendizagem à rede. Alias, com esta função de ativação, uma pequena alteração num *bias* ou num peso pode alterar todos os resultados da rede radicalmente uma vez que o seu output é binário. Por esse motivo foi introduzida a função *sigmoid* para função de ativação da rede neuronal. A função é representada no gráfico seguinte.

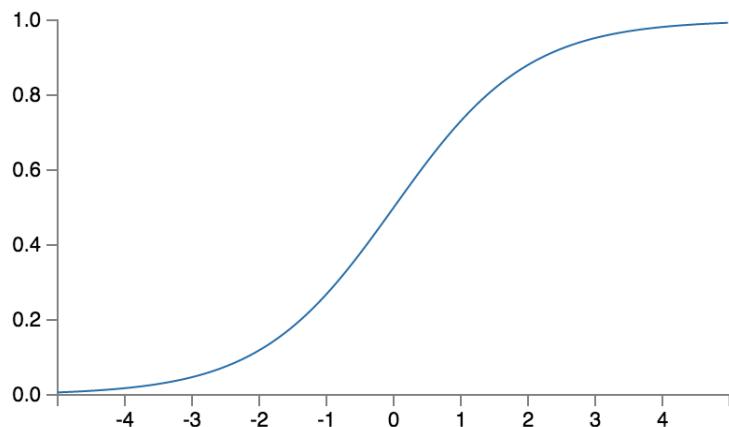


FIGURA 34 - FUNÇÃO SIGMOID.

Esta função introduz uma maior capacidade de aprendizagem à rede. O gráfico é representado pela seguinte função.

Em que o valor $-z$ é o resultado do neurónio. A seguinte imagem mostra uma rede neuronal utilizada para classificar imagens do *dataset* de *MNIST*.

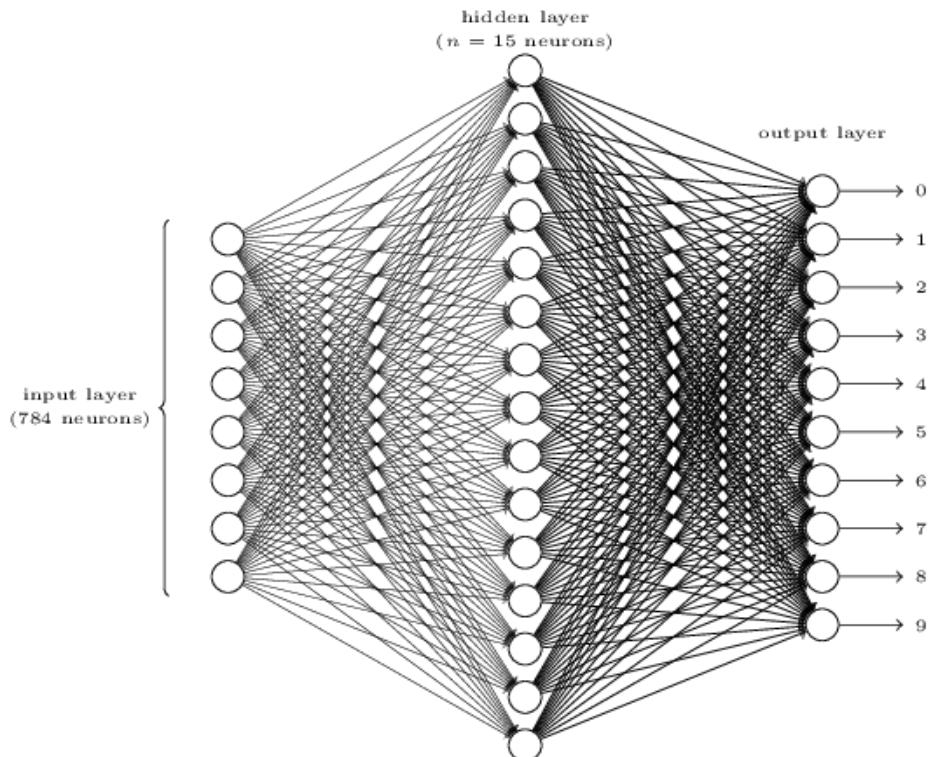


FIGURA 35 - REDE NEURONAL.

O número de neurônios de input (784) é dado neste caso pelo numero de pixéis da imagem e o numero de neurônios de output é dado pelos resultados expectados da rede. Numero de 0 a 9.

As redes neuronais convulsionais são semelhantes às redes neuronais convencionais no sentido em que utilizam pesos e *bias* passíveis de serem aprendidos. Porem, as redes neuronais ditas normais, não são escaláveis para imagens completas. No exemplo anterior vimos que a camada de input tinha 784 neurônios. Esse valor foi conseguido pela multiplicação de 28 por 28 pixéis da imagem de input numa escala de cinzentos. Se tivermos a mesma imagem na escala RGB teríamos de ter $28 \times 28 \times 3$ neurônios na camada de input o que indica que esta estrutura não é escalável a grandes imagens e poderia ficar bastante complexa. Não só, quando maior o numero de neurônios de input, significa que mais características a rede vai ter de aprender. Essas características são aprendidas na camada oculta. No exemplo anterior podemos fazer a analogia de cada neurônio da camada oculta como uma extração de uma característica da imagem a alto nível. Muito comum é haver a necessidade de extrair detalhes da imagem. Para isso seriam necessárias mais camadas ocultas que com o aumentar do nível da camada, aumentava também a abstração da imagem em si. Cada nível de camada oculta, extrairia uma característica da característica extraída no nível anterior. Várias camadas ocultas traria um problema grave para a aprendizagem da rede. O *gradient* que é o valor do erro do neurônio, tenderia a perder-se implicando que os neurônios mais perto da camada de input poderiam não aprender muito pouco ou nada.

As redes convulsionais foram desenvolvidas para ter em conta o processamento de imagem. Ao contrário das redes neuronais convencionais, a rede convulsional conta com camadas isoladas de três dimensões. Cada camada contém uma altura, uma largura e uma profundidade em que a profundidade representa o número de canais. A seguinte imagem compara as duas arquiteturas.

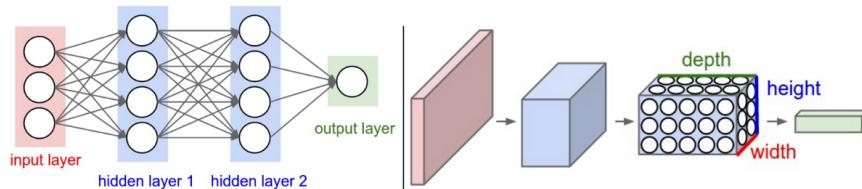


FIGURA 36 - REDE NEURONAL VS REDE CONVULSIONAL.

Também a maneira de ligar as diferentes camadas é diferente nas redes convulsionais. Ao contrário das redes convencionais, estas ligam os neurônios de uma camada a só uma pequena parte da camada anterior. Como camada final, a rede convulsional apresenta um vetor organizado somente na dimensão de profundidade.

Uma rede convulsional ou *CNN* tem uma lista de camadas que transformam a imagem de input numa imagem diferente através de uma função diferenciável. Existem algumas camadas diferentes:

- **Camada convulsional** – Consiste em vários filtros. A cada input, é feita uma convulsão produzindo um mapa do filtro (*kernel*) com a imagem
- **Pool** – Camadas inseridas entre camadas convulsionais. O objetivo é controlar o *overfitting* e reduzir o número de parâmetros e consequentemente de computação da rede. Também é óbvio que a camada de Pool entre outras coisas consiste num *kernel* de tamanho dois por dois que maximiza ou faz a média dos pixels. Assim consegue passar de uma imagem de 224 por 224 pixels para 112 por 112 pixels.
- **Sigmoid, Relu e softmax** – As funções de ativação mais utilizadas pelas CNN's e utilizadas no contexto deste trabalho.
- **Fully Connected** – Os neurônios nesta camada têm ligações a todos os neurônios das camadas anteriores como acontece nas redes neuronais convencionais que depois produzem um output consoante a função de ativação

A seguinte imagem mostra o processamento de uma imagem pela rede convulsional.

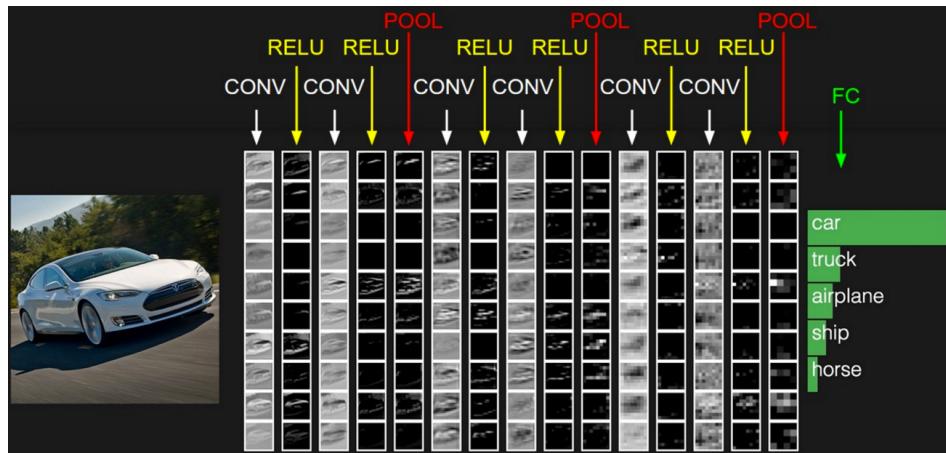


FIGURA 37 - DEMONSTRAÇÃO DAS CAMADAS DA CNN.

Podemos ver pela imagem, as características recolhidas por cada camada da rede neuronal convulsional.

3.2 Energias renováveis

Foram também estudadas, de modo a tornar o sistema o mais autossustentável possível dois métodos de extração de energia. A energia solar e a energia eólica, porém a energia eólica não tenho sido adaptada ao sistema. Nos próximos dois capítulos são descritos os métodos de extração deste tipo de fontes de energias renováveis.

3.2.1 Energia solar

A energia solar, provém do sol. O sol é capaz de gerar energia suficiente para alimentar todo o mundo e muito mais que pudéssemos utilizar. Para converter a luz e calor fornecidos pelo sol em energia, são utilizadas células solares.

As células solares são dispositivos elétricos que recolhem a luz solar e a transformam em energia. As células solares são muito pequeninas com forma octogonal que juntas formam módulos de células solares, que por sua vez se agrupam formando painéis solares.

Estas células, são células normais como as células de uma bateria de um carro, mas em vez de produzirem energia através de produtos químicos, produzem através da luz solar. Razão pela qual são muitas vezes chamadas de células fotovoltaicas.

Alem da energia solar, foi pesado como alternativa para quando o recurso é escasso, adaptar um sistema de energia eólica. No âmbito do presente relatório, o sistema não conta com energia eólica.

4 Análise de requisitos

Um fogo pode ser detetado através de vários métodos. Todos os fogos emitem luz infravermelha com várias intensidades do tom vermelho dependendo da temperatura. O fogo também emite fumo. Também este pode ser detetado, uma vez que é um indicador de fogo. Também o fumo muda com o calor emitido pelo fogo. Uma vez que o calor não pode ser detetado pelo olho humano, os componentes a serem estudados são o fumo e fogo. O problema que se prende com a deteção de fumo, é que não é fácil de definir em termos de processamento de imagem pois o fumo por ser leve é muito influenciado com a direção do vento descrito como movimento *Brownian*. É, portanto, importante treinar um modelo para detetar fumo o que se torna uma tarefa muito difícil. Ao momento, foram e continuam a ser explorados vários métodos para deteção de fogo e fumo. Vários são os métodos predominam na área da deteção de fogo. A utilização de mapas de cor como vermelhos, cor-de-laranja, deteção de cores perto do infravermelho deteção de limites, texture e formas. Todos os métodos têm em comum a necessidade de distinguir o fogo de regiões parecidas com o fogo pelas suas características. Do outro lado, o fumo uma vez que é um sinal de um eventual fogo, é um elemento de grande prioridade de deteção no processamento de imagem.

São utilizados vários métodos. Um dos mais populares, é a utilização de redes neurais para entender a forma arbitrária com que o fumo se transposta. O problema nesta abordagem, mesmo com técnicas mais elaboradas como redes neurais convulsionais, adaptadas para boa performance em imagem 2D, é que o processo de treino das mesmas é muito difícil levando a muitos erros dado a alta irregularidade do fumo e falta de indicadores característicos somente de fumo. Também aqui, o processo de treino do algoritmo influencia muito a performance da rede neuronal. A resolução comum é utilizar técnicas diferentes para a deteção de cada um dos elementos. Tipicamente o fogo é possível ser “ensinado” à máquina, ao contrário do fumo em que a sua deteção se prende apenas e só com processamento de imagem. Neste projeto, foi esta a abordagem utilizada. O fumo é detetado recorrendo a algoritmos de processamento de imagem, e o fogo numa primeira instância por algoritmos de processamento de imagem, e de seguida por algoritmos de *machine learning*.

Postos todos estes problemas e constatações, o requisito mais importante a ser explorado é a construção de uma rede que possibilite ao utilizador se determinada área contém fogo ou não. Desta forma, o utilizador consegue agir em concordância. Outro requisito fundamental é o envio de alertas para os utilizadores mediante o modelo probabilístico definido. Um *must have* da aplicação é também a capacidade de fazer *streaming* de vídeo. No entanto, o requisito mais importante que anda a par com a rede neuronal é o dataset reunido. Este requisito é o requisito

mais difícil não tecnicamente, mas é o requisito de mais difícil obtenção. O dataset deve conter os seguintes cenários:

- Fogo em vários estados de vida;
- Não fogo;
- Nevoeiro;
- Fumo;
- Por do sol;

Posto isto é possível reunir uma quantidade de requisitos para o funcionamento do sistema. Os requisitos a nível de tecnologias são os seguintes:

- Tecnologias de *machine learning* que sejam suportadas pelo *RaspberryPi*;
- Tecnologias de *cloud* que permitam o envio de notificações *push*(remotas) e fazer autenticação;
- Ter hardware que possibilite uma visão de 180 graus,
- Ser alimentado por um painel solar por intermédio de uma bateria;
- Fazer *streaming* de vídeo;
- Ter uma aplicação móvel para *Android*;

5 Sistema desenvolvido

Neste capítulo são descritos os componentes do sistema desenvolvido a nível do seu hardware e do seu software. Através da análise desta informação, conseguimos ter uma vista global sobre o projeto. Assim, vai ser no capítulo 5.1 apresentada a arquitetura do sistema e no capítulo 5.2 os módulos que compõem essa arquitetura e que foram desenvolvidos para funcionamento do mesmo.

5.1 Arquitetura do sistema desenvolvido

A arquitetura é um gráfico importante do sistema, no sentido que dá uma visão de alto nível sobre o mesmo. A seguinte figura ilustra a arquitetura do sistema.

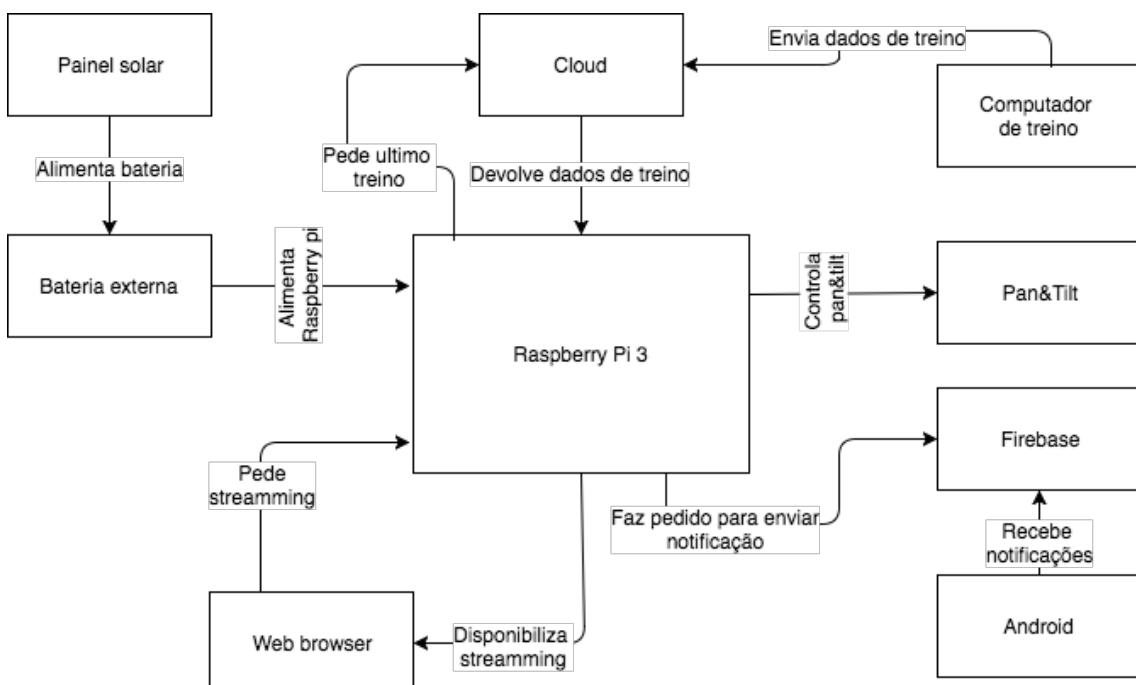


FIGURA 38 - ARQUITETURA DE HARDWARE.

Na figura podemos ver como o hardware do sistema se interliga. O *RaspberryPi* representa o nó central que é alimentado por uma bateria que por sua vez é carregada com energia proveniente de painéis solares. O *RaspberryPi* implementa um protocolo para comunicar com um *Arduino* de modo a ter a função de *pan&tilt* da câmara. A razão de não ser o *RaspberryPi* diretamente a controlar os motores de *pan&tilt* é somente a performance. Uma vez que o sistema instalado no *RaspberryPi* não é um sistema de tempo real, nem o mesmo conta com um *clock*, o *RaspberryPi*, não é o melhor para tarefas críticas. Assim, separando estes dois componentes com um *Arduino*, estamos a introduzir um microcontrolador de tempo real e ao mesmo tempo, o *RaspberryPi* fica com menos uma tarefa. Também o *RaspberryPi* é responsável pela implementação de uma camara que recolhe imagens a serem classificadas. As imagens

encontram-se disponíveis através de um serviço web. O classificador é responsável ainda por fazer o download de novos dados de treino da *cloud* e por enviar notificação sempre que deteta fogo através do serviço *Firebase*.

5.2 Módulos do Sistema

Neste subcapítulo são descritos os vários módulos que compõem o sistema. Neste capítulo, estão ilustrados alguns dos mais importantes algoritmos, a funcionalidade de cada módulo e um guia para posterior utilização.

5.2.1 Implementação do dataset

O passo inicial no desenvolvimento de qualquer sistema relacionado com machine learning é a construção de um dataset.

O dataset deve ter como já referido três fases. Fase de treino, fase de validação e fase de testes. Deve também ser dividido tendo em conta regras conhecidas. O dataset é o conjunto de dados que representam as diferentes categorias que queremos que a rede neuronal distinga. Neste caso, o dataset obtido representa duas categorias

Como referido anteriormente, o projeto foi desenvolvido com o recurso a algoritmos de redes neurais, mais especificamente *CNN's*. As *CNN's* são compostas por várias fases até chegar a um produto final.

Esta fase é um grande desafio especialmente nas *CNN's* pois a anatomia de uma *CNN's* é a geração de vários filtros através de convulsões e que vão tendo os seus valores ajustados durante uma fase de treino da rede como já explicado. Isto faz com que, a *CNN* precise de um grande volume de dados para ser treinada de forma eficiente.

Com esse objetivo neste projeto foram desenvolvidos vários scripts para ajudar na construção do dataset. A seguinte imagem explica o fluxo de obtenção e organização do dataset com o objetivo de ser consumido pela rede neuronal.

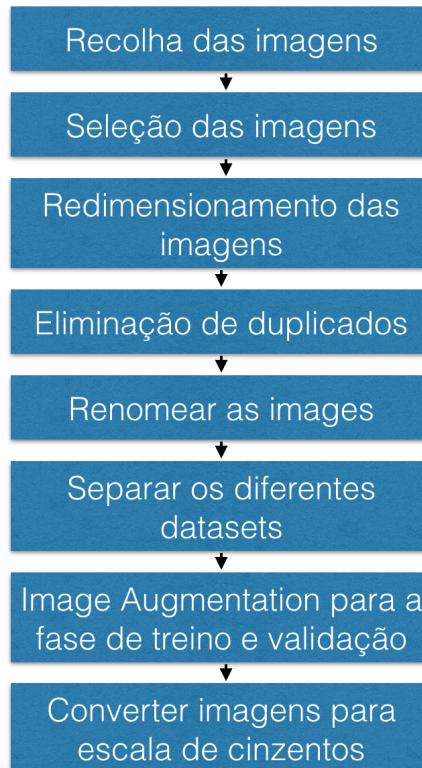


FIGURA 39 - PROCESSO DE TRATAMENTO DO DATASET.

DOWNLOAD DE IMAGES

O script permite fazer download de imagens do site *forrestryimages* e utiliza a *API* (Application programming interface) *Pixabay* para o mesmo efeito. O script aceita quatro argumentos de input obrigatórios:

O script aceita como parâmetros uma query, uma pasta de destino para guardar as imagens, e um prefixo da image Na sua essência, o script é bastante simples.

A pesquisa ao site *forrestryimages* é na prática uma pesquisa pelos elementos *HTML*. Numa primeira fase, faz uma query ao servidor sobre primeira página de imagens. Cada página contém 200 imagens. Nesta fase, é procurado o numero total de imagens e dividido por 200 para obter o numero total de páginas e assim enumerar e mudar o numero de página no url. Seguidamente e tendo em conta a página em questão, procura na página html por *tag's img*. A essas tag's é extraído o *url* da imagem contido no atributo *src*. O próximo passo é fazer o download da imagem e guarda-la no diretório passado como argumento ao script.

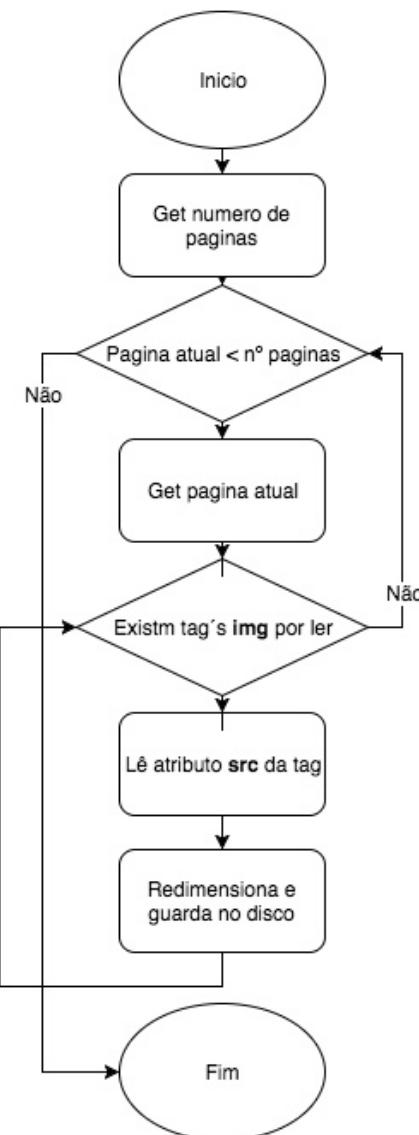


FIGURA 40 - FLUXOGRAMA DE OBTENÇÃO DE IMAGENS.

No mesmo script, é utilizada a *API* Pixabay para fazer download de imagens da base de dados. A *API* disponibiliza um serviço *REST*(Representational state transfer). Da mesma forma, mas agora com um pedido *GET* à *API* da Pixabay em primeiro lugar descobrimos o numero de páginas existem para dada pesquisa e iteramos ao numero de páginas com diferentes pedidos à *API*. Cada pedido a uma página devolve da *API* um dicionário com o url da imagem que depois é utilizado para guarda a imagem em disco.

Em todo o processo de download de imagens em qualquer serviço é importante salientar que:

- A imagem é guardada com um prefixo que corresponde à sua categoria;
- A imagem é guardada com um numero decimal aleatório de modo a não haverem nomes repetidos e evitar espaçamentos no meio do nome;

A seguinte imagem mostra a chamada do script e o download de imagens do site forestry.

```
[Diogos-MBP:utils diogoaleixo$ python images_downloader.py -q forrest -d Downloads/ -p forrest -r 32
[INFO] Forestry images
[INFO] Total of 16 images in 0 pages
-----
[INFO] Getting page 0
-----
[INFO] Downloaded image num 2 from page 0 and saved to Downloads//forrest/forrest_6441.87316674.jpg
[INFO] Downloaded image num 3 from page 0 and saved to Downloads//forrest/forrest_370.748606434.jpg
[INFO] Downloaded image num 4 from page 0 and saved to Downloads//forrest/forrest_1250.15752937.jpg
[INFO] Downloaded image num 5 from page 0 and saved to Downloads//forrest/forrest_9612.49139952.jpg
[INFO] Downloaded image num 6 from page 0 and saved to Downloads//forrest/forrest_7794.48610402.jpg
[INFO] Downloaded image num 7 from page 0 and saved to Downloads//forrest/forrest_4457.31718166.jpg
[INFO] Downloaded image num 8 from page 0 and saved to Downloads//forrest/forrest_4063.94482799.jpg
[INFO] Downloaded image num 9 from page 0 and saved to Downloads//forrest/forrest_4939.37480282.jpg
[INFO] Downloaded image num 10 from page 0 and saved to Downloads//forrest/forrest_3453.61992772.jpg
[INFO] Downloaded image num 11 from page 0 and saved to Downloads//forrest/forrest_8021.81361218.jpg
```

FIGURA 41 - OUTPUT SCRIPT DE DOWNLOAD DE IMAGENS.

Na imagem anterior podemos ver como o script é chamado, os parâmetros utilizados e o output fornecido pelo script. Conseguimos ver para cada página o numero de página que está a fazer download, onde está a salvar cada imagem e com que nome.

A seguinte imagem mostra ainda o download com a mesma query de imagens do serviço *Pixabay*.

```
-----
[INFO] Getting page 0
-----
https://pixabay.com/api/?key=3069392-4e571367e1943bff61c178bc0&q=forrest&per_page=200&pages&page=1
[INFO] Downloaded image num 0 and saved to Downloads//forrest/forrest_944.549885917.jpg
[INFO] Downloaded image num 1 and saved to Downloads//forrest/forrest_4322.79890453.jpg
[INFO] Downloaded image num 2 and saved to Downloads//forrest/forrest_6495.21039019.jpg
[INFO] Downloaded image num 3 and saved to Downloads//forrest/forrest_3555.66768758.jpg
[INFO] Downloaded image num 4 and saved to Downloads//forrest/forrest_8637.74607202.jpg
[INFO] Downloaded image num 5 and saved to Downloads//forrest/forrest_1825.08217563.jpg
[INFO] Downloaded image num 6 and saved to Downloads//forrest/forrest_1794.27871651.jpg
```

FIGURA 42 - OUTPUT SCRIPT DE DOWNLOAD DE IMAGENS.

A informação dada quando está a fazer download de imagens do *Pixabay* é sensivelmente a mesma.

Seleção das imagens

Este é o passo mais minucioso de todo o processo. Depois de ter sido feito o download das imagens pelos vários meios foi constatado que as procura retornaram imagens com o mesmo significado da procura, mas com um conteúdo desenquadrado com o pretendido. A imagem seguinte mostra o exemplo de uma imagem que retornada pela pesquisa de “wildfire” que não se enquadrava no pretendido.



FIGURA 43 - EXEMPLO DE MAU INPUT PARA O DATASET.

Na imagem anterior vemos claramente uma situação em que foi retornada uma imagem que em nada ajuda o classificador pois o que é pretendido é classificar fogo, não uma situação em que o fogo já está extinto. Existem outras pesquisas que retornaram resultados errados, como por exemplo “forrest” que retornou algumas imagens do filme forrest gump.

Redimensionamento das imagens

Aquando da passagem do dataset para o classificador, as imagens têm de ter um tamanho uniforme.

Na prática, quando falamos na primeira camada consensual, estamos a falar de uma matriz de n colunas por m linhas. Essa matriz tem um tamanho fixo, como tal, todas as imagens deverão ter dimensões iguais.

As dimensões a escolher dependem do sistema que dispomos para o treino e da quantidade de dados que temos. Regra geral, quantos mais dados dispomos, maior pode ser o tamanho das imagens do dataset. Esta razão deve-se à capacidade de reunir diversas ativações para classificar determinada classe. Neste caso foi escolhida uma dimensão de 64 por 64 pixéis devido à limitação do hardware. Em contas rápidas, só pela imagem, a rede conta com 4096 parâmetros de entrada. Uma imagem deste tamanho e tendo em conta que temos 32 filtros na primeira camada, vai ocupar sensivelmente 0.125 MegaBytes em memória. Em primeira instancia não parece critico este tamanho. O problema é que em tempo de treino todas a imagens de treino

estão alocadas em memória. Na prática este valor escala rapidamente para as centenas de megabytes.

No terminal com o comando `python resizeImagesInFolder.py -d ~/Desktop/dataset/ -s 64` chamamos o script para redimensionar as imagens. O argumento “-d” indica a diretoria das imagens a serem redimensionadas e o “-s” indica qual o tamanho final das imagens.

```
(/Users/diogoaleixo/anaconda/envs/python27) bash-3.2$ python resizeImagesInFolder.py -d ~/Desktop/dataset/ -s 64
Using TensorFlow backend.
[UTILS] Will resize images in /Users/diogoaleixo/Desktop/dataset/ with 64 pixels per w*h
[UTILS] Load image /Users/diogoaleixo/Desktop/dataset/fire-11.7227995062.png to resize
[UTILS] Load image /Users/diogoaleixo/Desktop/dataset/fire-13.3110033796.png to resize
[UTILS] Load image /Users/diogoaleixo/Desktop/dataset/fire-26.5997160611.png to resize
[UTILS] Load image /Users/diogoaleixo/Desktop/dataset/fire-45.7396875153.png to resize
[UTILS] Load image /Users/diogoaleixo/Desktop/dataset/fire-63.1076686855.png to resize
[UTILS] Load image /Users/diogoaleixo/Desktop/dataset/fire-7.48271836353.png to resize
[UTILS] Resized 6 images
```

FIGURA 44 - OUTPUT SCRIPT DE REDIMENSIONAMENTO DAS IMAGENS.

Imagens duplicadas

Alem do problema de imagens que não se enquadram no dataset pretendido, a pesquisa por imagens devolve muitas vezes imagens iguais. De notar que o igual quer dizer mesmo igual. Igual ao nível do pixel. Caso a imagem seja só semelhante, as características que o classificador vai conseguir aprender serão diferentes.

Este passo é tomado depois do redimensionamento das imagens propositadamente. Aquando da obtenção do dataset notei que a mesma imagem era utilizada em sites diferentes com dimensões diferentes. Antes de verificar igualdades, metemos tudo com a mesma dimensão sem nos preocuparmos com o *aspect ratio*.

A técnica neste caso para comparar duas imagens foi a extração de histogramas dos três canais (*RGB*). O histograma representa num gráfico a relação entre a quantidade de pixels e determinado valor (0 a 255). Na imagem seguinte é mostrado o histograma de duas imagens diferentes.

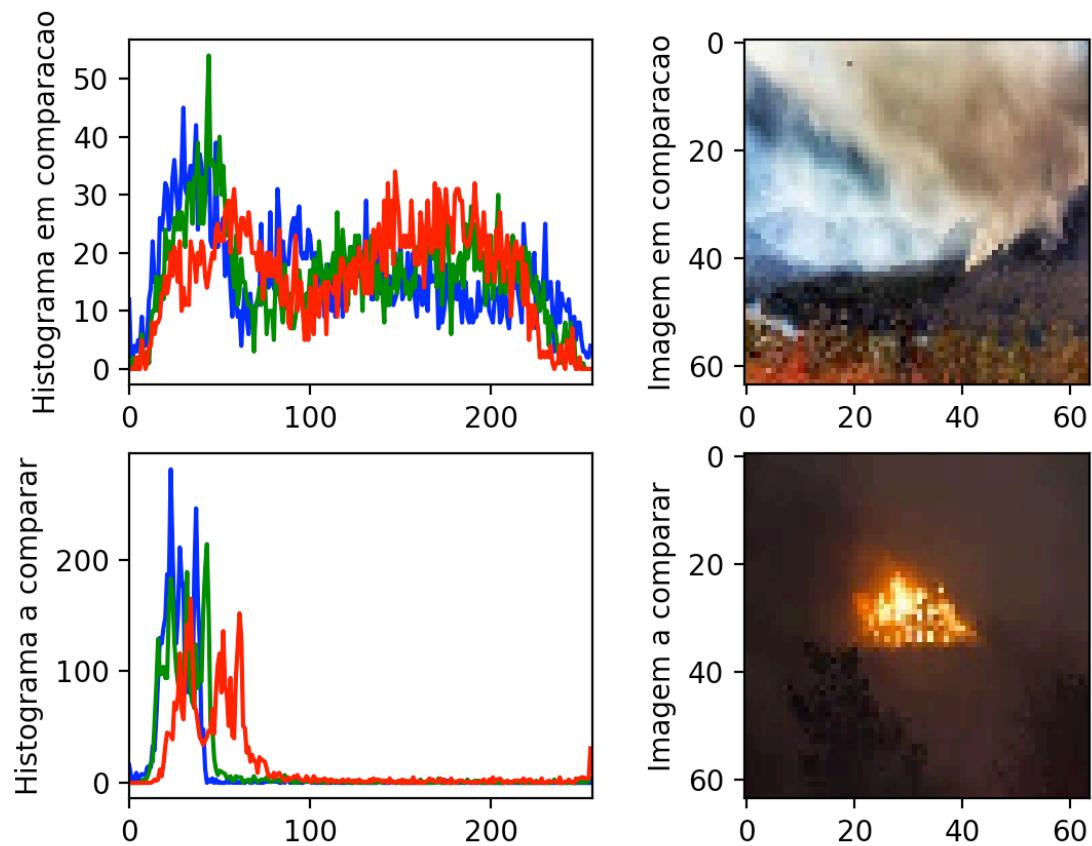


FIGURA 45 - HISTOGRAMAS DIFERENTES.

Estes dois histogramas demonstram na perfeição o significado. Por um lado, vemos uma imagem mais clara que teoricamente contem valores de pixels mais elevados. Por outro lado, temos uma imagem mais escura em que é esperado um histograma com valores dos pixels reduzidos. O método utilizado para a comparação de dois histogramas foi a distância euclidiana entre dois pontos. O script desenvolvido elimina imagens que tenham dois histogramas iguais como mostra a seguinte figura.

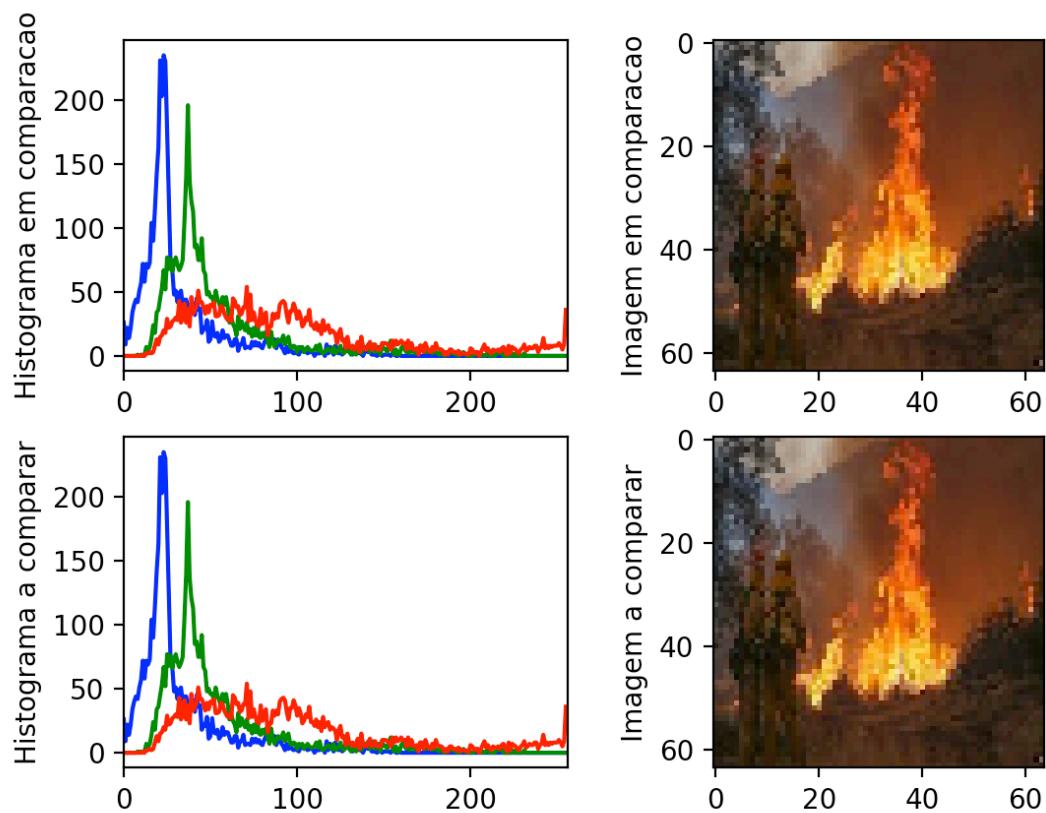


FIGURA 46 - HISTOGRAMAS IGUAIS.

O algoritmo para a comparação das imagens é relativamente simples uma vez que a matemática já se encontra programada no biblioteca *OpenCv*.

-
1. Definir lista de cores
 2. Definir listas para os histogramas e histogramas2
 3. Para imagemOriginal em diretoria
 4. Imagem = Ler a imagemOriginal
 5. Canais = Separar os canais da imagem
 6. Para canal em Canais
 - i. Calcular histograma do canal
 - b. Fim para
 - c. Para imagemDestino em diretoria
 - i. Se a imagemDestino for diferente da imagemOriginal
 1. Imagem2 = ler a ImagemDestino
 2. Canais2 = separar os canais da imagem
 3. Para canal em canais2
 - a. Adicionar histograma do canal à lista de histogramas2
 4. Fim para
 5. Para histograma no conjunto das listas dos histogramas
 - a. Se comparação do histograma for 0
 - i. Incrementar canaisIguais
 - ii. Se canaisIguais for 3
 1. Remover imagemDestino
 2. Guarda histogramas 3D na diretoria
 - iii. Fim se
 - b. Fim se
 6. Fim para
 - ii. Fim se
 - d. Fim para
 7. Fim para

ALGORITMO 1 - COMPARAÇÃO DE IMAGENS

No terminal com o comando `python deleteDuplicateImages.py -d ~/Desktop/dataset/` em que o argumento “-d” é o diretório das imagens RGB obtemos a informação visível na seguinte imagem.

```
[UTILS] Checking for duplicate images in path /Users/diogoaleixo/Desktop/dataset/fire-7.48271836353.png and /Users/diogoaleixo/Desktop/dataset/fire-7.48271836353copy.png
[DELETE DUPLICATE IMAGES] Equal images /Users/diogoaleixo/Desktop/dataset/fire-7.48271836353.png /Users/diogoaleixo/Desktop/dataset/fire-7.48271836353copy.png
[UTILS] Folder had 9 images and now has 8. 1 were deleted
```

FIGURA 47 - OUTPUT SCRIPT ELIMINA DUPLICADOS.

Na imagem anterior vemos que foi eliminada uma imagem da pasta passada como parâmetro.

Renomear as imagens

Apesar de todos os passos anteriores serem importantes e contribuirem para o objetivo de treinar um classificador com boa capacidade de generalização para outros inputs, este passo é provavelmente o mais importante.

Sem passos como, redimensionar as imagens ou eliminar duplicados, o sistema dá error no primeiro caso, e no segundo pode sofrer de *overfitting*. Se der error, o programador consegue resolver, se sofrer de *overfitting* também consegue analizar a curva de aprendizagem, chegar a essa conclusão e resolver.

No caso de as imagens não estarem com nomes corretos, o problema pode ser outro pois é partir destes nomes que o classificador vai fazer uma separação das classes em tempo de treino, validação e teste. Caso os nomes não estejam bem, o classificador não devolve nenhum erro. Simplesmente classifica mal, o que pode ser difícil de resolver.

Neste caso o nome tem a seguinte sintaxe “*keyword-random_number.png*” em que keyword é fire ou forrest e o numero aleatório é um numero decimal que permite minimizar a hipótese de nomes iguais. A *keyword* vai ser utilizada mais tarde para preencher uma matriz que corresponde com cada linha a corresponder aos piteis de determinada imagem e a sua classificação.

No terminal, ao correr o comando *python renameImagesInDir.py -d ~/Desktop/dataset/-p test*, em que o argumento “d” é o directorio onde estão as imagens a renomear e o argumento “p” é o prefixo ou *keyword* da imagem, é devolvido o seguinte output.

```
[UTILS] Renaming image fire-11.7227995052.png to /Users/diogoaleixo/Desktop/dataset/teste-2300.69436083.png
[UTILS] Renaming image fire-13.3110033796.png to /Users/diogoaleixo/Desktop/dataset/teste-9155.50393297.png
[UTILS] Renaming image fire-26.5987168611.png to /Users/diogoaleixo/Desktop/dataset/teste-6028.7891856.png
[UTILS] Renaming image fire-45.7396875153.png to /Users/diogoaleixo/Desktop/dataset/teste-2601.44304707.png
[UTILS] Renaming image fire-63.107660685.png to /Users/diogoaleixo/Desktop/dataset/teste-954.143346911.png
[UTILS] Renaming image fire-63.107660685d.png to /Users/diogoaleixo/Desktop/dataset/teste-1804.78723647.png
[UTILS] Renaming image fire-7.48271836353.png to /Users/diogoaleixo/Desktop/dataset/teste-7407.02884293.png
```

FIGURA 48 - OUTPUT SCRIPT RENOMEAR IMAGENS.

Podemos ver neste exemplo que as imagens ficarão agora com o prefixo “teste” indicado no argumento “p” do script

Data Augmentation

Apesar dos esforços no sentido da obtenção de um maior dataset, os mesmos não foram suficientes devido a vários factores. No entanto, existem outras técnicas que podem ser utilizadas para aumentar o dataset com as mesmas imagens sem as repetir.

Data augmentation é uma técnica composta por várias técnicas que visa aumentar o numero de imagens num dataset utilizando o mesmo como input. Como exemplo, do que *data augmentation* faz, esta técnica pode utilizar rotações na imagem e assim conseguir várias imagens diferentes do original. Esta técnica vem tentar resolver o problema de datasets pequenos, fazendo com que a mesma imagem contenha representações iguais de formas diferentes. Assim, a rede consegue analisar para o mesmo exemplo mais imagens e não correr o risco de *overfitting* pois têm diferentes representações.

Neste exemplo em concreto são geradas imagens com as seguintes características:

- Rotações aleatórias;
- Flip's;
- Espelhamento de imagem;
- Zooming de imagem;
- Normalização de cores da imagem;

Como exemplo, a imagem seguinte mostra um conjunto de imagens que sofreram transformações características desta técnica.

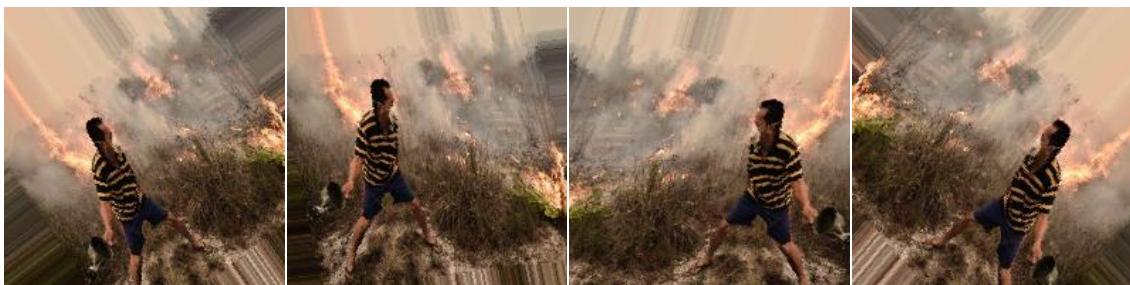


FIGURA 49 - EXEMPLO DE DATA AUGMENTATION.

5.2.2 Firebase e Google Drive

O projeto incorpora características de computação distribuída e de alarmes remotos. Para o efeito, é utilizada comunicação com o *Firebase* e com o *Google Drive*. Estes componentes permitem Autenticação na *cloud*, notificações *push* e armazenamento de ficheiros.

Firebase

O *Firebase* é uma ferramenta disponibilizada pela *Google* que entre várias opções, o envio de notificação remotas para clientes *Android* e *iOS*, a autenticação de clientes, entre outros serviços

Disponibilizado de forma gratuita o *Firebase* é construído em cima do anterior *Google Cloud Messaging* com novas funcionalidades.

Neste projeto foi utilizada a vertente de notificações e *cloud messaging* do *Google Firebase* e autenticação. Tanto as notificações como o *cloud messaging* são utilizados para permitir o envio de notificações remotas para a aplicação móvel. Essas notificações são enviadas no caso do sistema *Android* por uma interface *REST* disponibilizada pela *Google*. De seguida a notificação é enviada para os clientes previamente registados no serviço. No caso do *iOS*, a notificação é enviada através da *API REST* disponibilizada e de seguida enviada para os servidores de notificações da *Apple* (*APNS*) que se encarregam por entregar a mensagem às aplicações corretas. A imagem seguinte demonstra a arquitetura básica do sistema descrito.



FIGURA 50 - ARQUITETURA GOOGLE FIREBASE.

A configuração do *Firebase* é relativamente simples. O serviço disponibiliza um *endpoint* para o qual devemos enviar uma *string JSON*(JavaScript object notation) formatada com dados como, tipo da notificações, clientes a receber, avisos sonoros entre outros. Essa *string*, é no contexto do sistema desenvolvido, elaborada sempre que o alarme de deteção de fogo excede determinados limites.

No âmbito do projeto, o *Firebase* é ainda utilizado para fazer a autenticação de clientes da aplicação móvel. Desta maneira, torna possível adicionar rapidamente segurança à aplicação. O seguinte diagrama mostra a interação do serviço *Firebase* com o sistema desenvolvido.

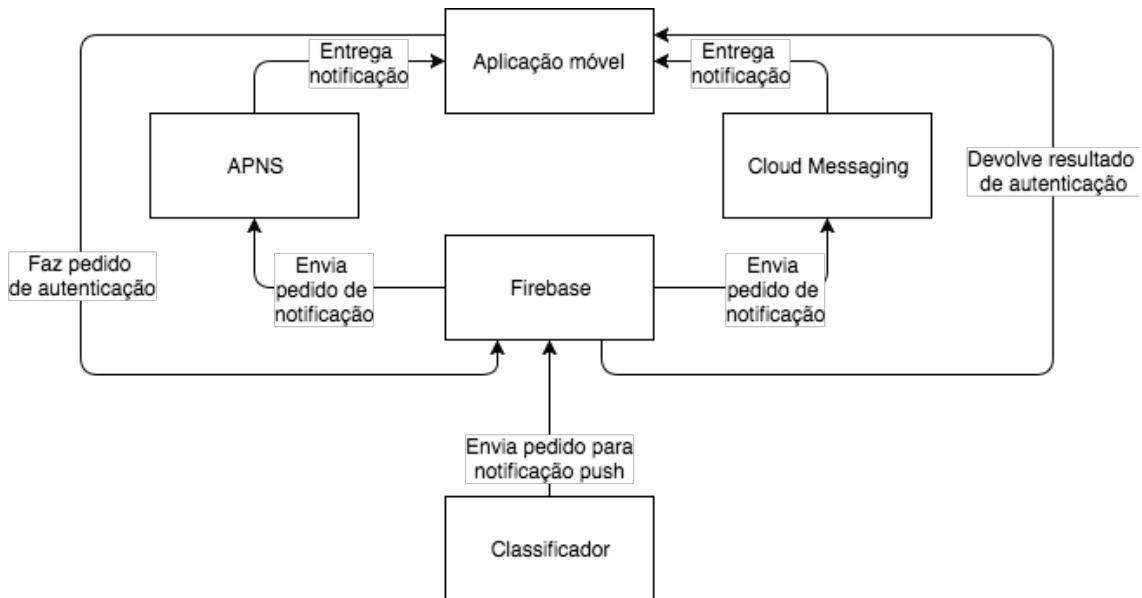


FIGURA 51 - DIAGRAMA DE INTEGRAÇÃO DO SISTEMA COM O FIREBASE

Google drive

O *Google Drive* é a mais bem aceite tecnologia de Storage na cloud. O sistema utiliza o *Google Drive* para armazenar ficheiros e podes distribuir facilmente sem intervenção humana.

Como explicado no diagrama de hardware, o sistema desenvolvido entre outras componentes é constituído por uma máquina de treino. O objetivo desta máquina de treino é simplesmente treinar noas arquiteturas de redes neurais e distribui-las pelos clientes. A seguinte figura ilustra o conceito.

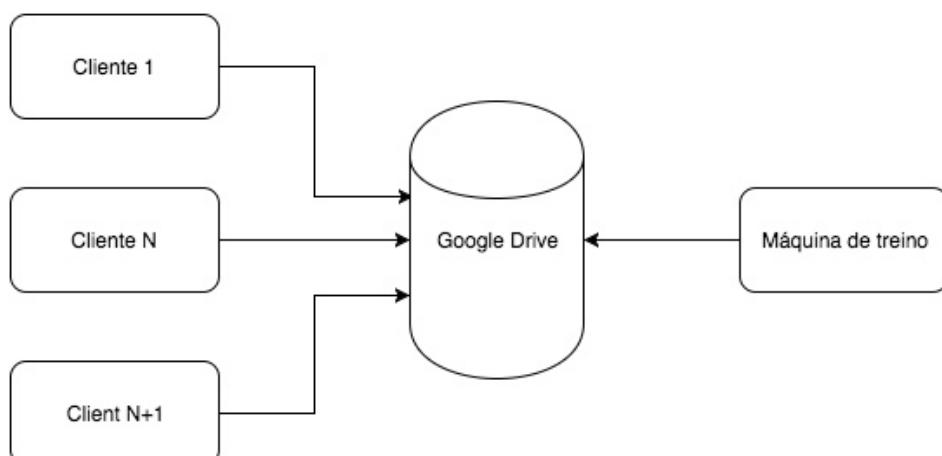


FIGURA 52 - ARQUITETURA TREINO REMOTO.

Uma máquina de treino pode ter vários clientes. A mesma, faz o *upload* seguidamente ao termo do treino da rede com os seus novos valores para determinada conta no *Google Drive*. Quando um cliente reinicia a aplicação de deteção de fogos, é lançado um processo responsável por fazer o download do ultimo treino disponível na mesma conta do *Google Drive*. Desta maneira, é possível garantir que facilmente os clientes usufruem do melhor método de deteção e que a fiabilidade da rede pode facilmente ser aumentada.

5.2.3 Classificador

A base deste trabalho é, como já referido uma rede neuronal. Esta rede neuronal tem uma arquitetura diferente das redes neuronais artificiais. É uma rede construída para utilização em visão computacional.

Este capítulo não tem como objetivo reforçar a explicação já dada em capítulos anteriores sobre *CNN's* mas sim explicar o desenvolvimento da mesma no contexto do projeto.

Depois de recolhido o dataset, o próximo passo para a definição de uma *CNN* é a sua arquitetura. A arquitetura diz a alto nível qual o grau de trabalho que a rede vai ter para classificar determinado input. No entanto não é proporcional o tamanho da arquitetura e sua complexidade ao grau de eficácia da rede por problemas como *overfitting*. A seguinte figura ilustra a arquitetura da rede utilizada no projeto.

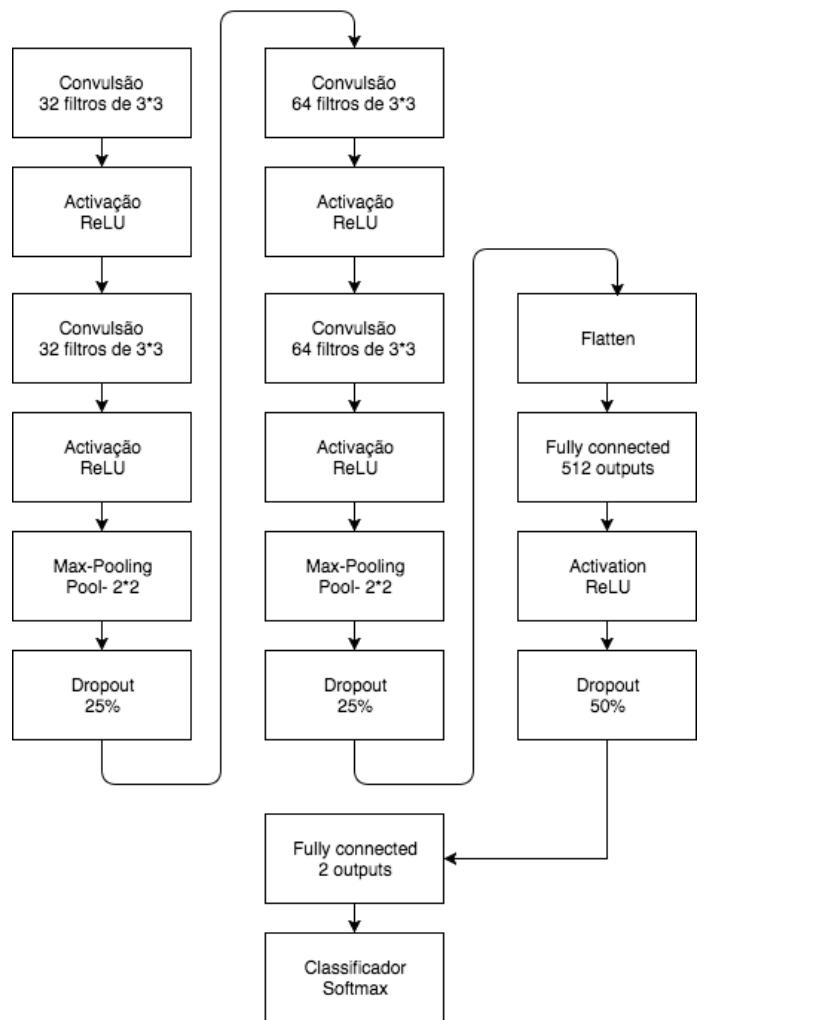


FIGURA 53 - ARQUITETURA DE REDE DO SISTEMA.

Na imagem em cima, é ilustrada a arquitetura parâmetros da rede. Podemos ver que a arquitetura é composta essencialmente por dois grandes blocos de convulsões, um grande bloco de uma rede neuronal artificial e um classificador *softmax*.

Bloco I convulsões

Este bloco recebe o input do classificador. Neste caso recebe uma imagem em escala de cinzentos.

Na prática, não é uma imagem que o bloco recebe, é uma matriz de uma dimensão de tamanho 64 por 64. A essa matriz vão ser aplicadas convulsões de 3 por 3. Essas convulsões estão identificadas na imagem como filtros. São criados 32 filtros. A seguinte imagem representa o primeiro grande bloco.

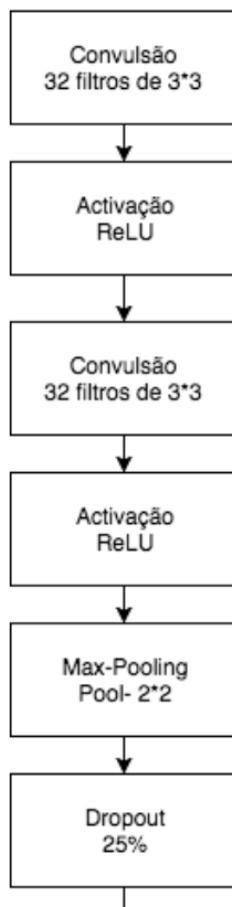


FIGURA 54 - PRIMEIRO BLOCO.

O primeiro bloco vai criar 32 matrizes de 3×3 resultantes de convulsões com a matriz original. De seguida, é aplicada uma função de ativação que vai definir se determinado neurónio é ou não excitado com os valores de input. O próximo passo é criar mais 32 matrizes de convulsões a serem aplicadas ao que a função de ativação ativou (matrizes que representam filtros de imagens como veremos mais tarde). Essas matrizes vão mais uma vez ser passadas por uma função de ativação *ReLU*. De seguida é utilizada uma camada de *Max Pooling*. O objetivo desta camada é reduzir a dimensão do input retendo as melhores características. Esta camada calcula na mesma filtro, mas não serão filtros aprendidos por pesos ou bias atualizados a cada época. São filtros que serão calculados através de uma fusão de maximização do input. Neste exemplo é aplicada uma janela de 2×2 , se o input for um array de [132,1,255,12] o output será 255.

O ultimo passo deste primeiro grande bloco é a aplicação de uma camada de *dropout*. Esta camada com uma taxa de 25% vai literalmente eliminar 25% das ligações da camada de ativação anterior. Esta camada é utilizada para prevenir que a rede se adapte somente aos dados que está a visualizar. A camada de *dropout* faz isso eliminando 25% das suas ligações de input o que força uma aprendizagem redundante e traz maiores certezas para o classificador. Esta técnica faz com que seja necessário treinar o classificador por mais épocas.

Bloco 2 convulsões

É normal depois de reduzir a dimensionalidade dos dados, aumentar o numero de extrações sobre os mesmos. A seguinte imagem mostra o segundo grande bloco.

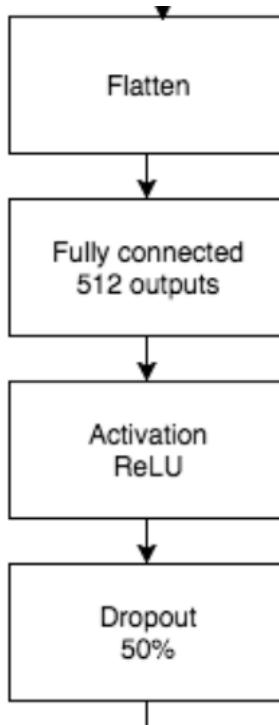


FIGURA 55 - SEGUNDO BLOCO.

Este bloco é um espelho do primeiro mas recolhe 64 matrizes a partir de uma matriz de input dada pela anterior camada de dropout.

Bloco 3 rede neuronal artificial

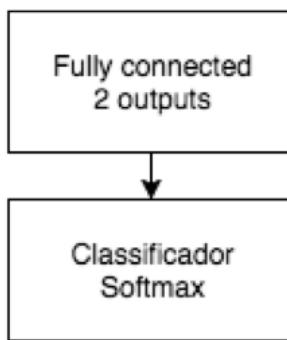
O ultimo grande bloco é uma rede neuronal artificial comum. Uma rede normal onde todos os neurônios estão interligados entre si. A seguinte imagem mostra o terceiro grande bloco.

**FIGURA 56 - BLOCO FINAL.**

O resultado final do bloco anterior é uma matriz e é passado à camada flatten, a primeira camada deste grande bloco. A camada flatten, pega numa matriz de duas dimensões e transforma em uma dimensão para ser passado como input a uma rede neuronal artificial com 512 neurônios de output. O resultado desta rede neuronal pode ser considerado como um feature vector. Este feature vector representa um determinado objeto, neste caso a imagem de input ao classificador. De seguida, o vetor é passado para uma função de ativação *ReLU* e é-lhe aplicada uma camada de *dropout* de 50%.

Classificador

A seguinte imagem representa o bloco responsável por classificar a imagem.

**FIGURA 57 – CLASSIFICADOR.**

Por fim, existe uma rede neuronal com imputa igual ao output anterior (512) e um output de somente dois neurônios. Estes dois neurônios são responsáveis por identificar uma das duas classes classificadas no processo. O output da rede é depois passado para um classificador *softmax*. Nesta fase, já existe uma classificação sobre a classe correta, esta função de atuação/classificador, é responsável por atribuir uma percentagem de classificação para depois ser utilizada no *backend* para enviar alertas.

Depois da arquitetura definida é necessário proceder à divisão do dataset. Como explicado em capítulos anteriores, o dataset tem de ser dividido em três fases. Fase de treino, fase de validação e fase de teste.

Neste projeto, a fase de teste são 20% do dataset total. O dataset total é de 18088 imagens logo o dataset de teste é de 3618 imagens. Este dataset não é tocado até à fase final de maneira a não influenciar diretamente ou indiretamente o treino.

Com o dataset de teste calculado, sobram 14470 imagens para serem divididas entre treino e validação. A divisão para o dataset de validação segue a mesma regra dos 20%. Concluindo o dataset fica com a seguinte divisão.

- Treino - 11576;
- Validação - 2894;
- Tese - 3618;

Embora tenhamos utilizado a regra dos 20%, o dataset de validação não conta com 20% do dataset global, mas sim do dataset para treino e validação. A seguinte imagem mostra um gráfico circular da divisão do dataset.

■ Treino ■ Validação ■ Teste

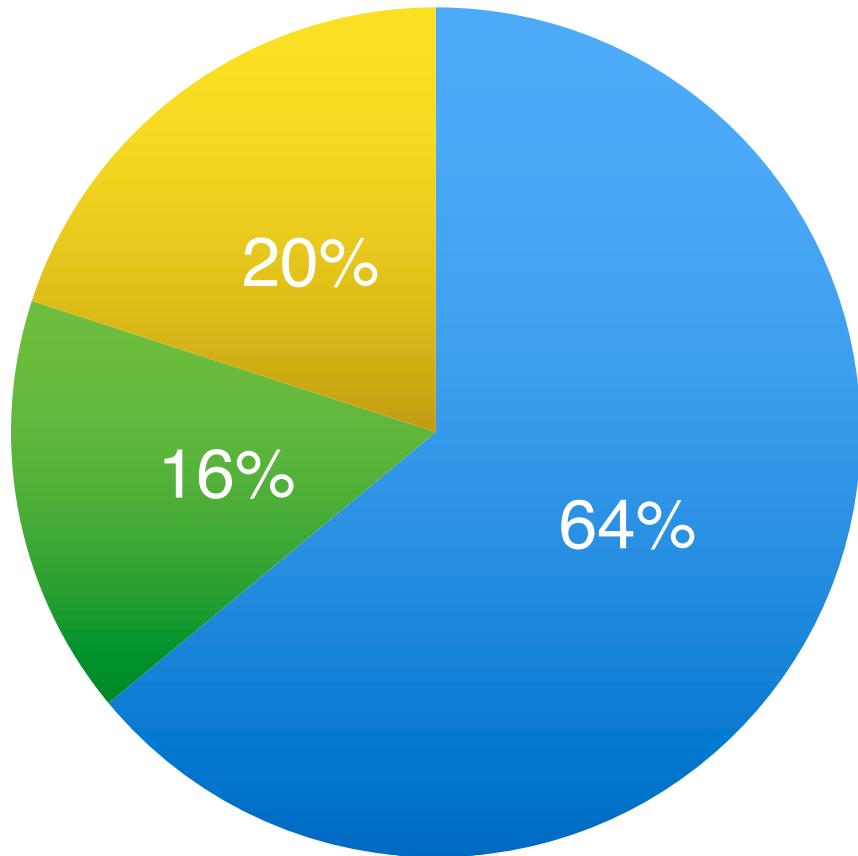


FIGURA 58 - DIVISÃO DO DATASET.

Depois de definida a arquitetura e os diferentes datasets, é preciso converter as imagens para o formato necessário para a *CNN*. Além da conversão existe um processamento prévio a fazer. A seguinte imagem mostra o fluxo dos passos para obter um formato otimizado para ser dado como imputa ao classificador.

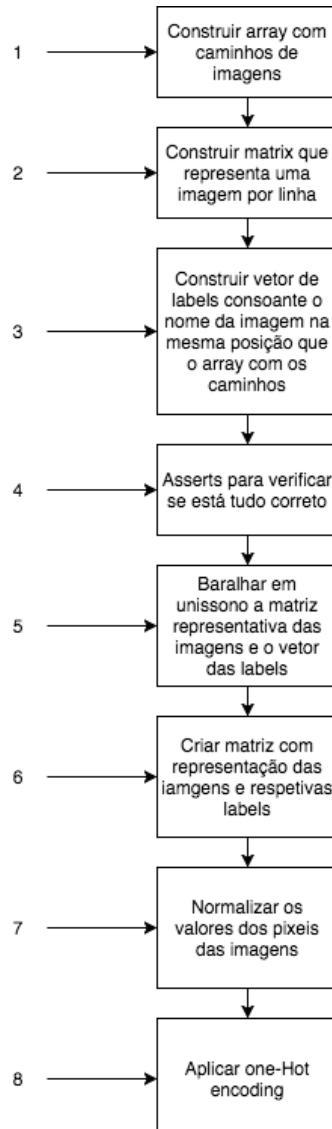


FIGURA 59 - PASSOS PARA CARREGAR AS IMAGENS.

A imagem anterior mostra 8 passos ordenados por ordem. Os 8 passos, constituem os passos necessários para carregar uma representação das imagens num formato legível para a rede:

- O primeiro passo é obtenção num *array* dos caminhos para todas as imagens.
- De seguida, é construído uma matriz de n linhas por 64*64 colunas com os valores de cada pixel da imagem por linha;
- Para a classificação, é preciso associar a cada imagem uma label que será a classificação final. Neste passo utilizamos um passo da preparação do dataset que tem como fim alterar o nome de todas as imagens do dataset. Neste caso, é feito um split do nome e verificado se não é um ficheiro de sistema. O primeiro índice do *array* resultante do *split* corresponderá à palavra “fire” ou “forrest” que determinam a categoria da imagem. 0 ou 1 respetivamente. Esse vetor tem a mesma ordem que o *array* dos caminhos das imagens.

- De seguida são feitos alguns *asserts* para validar se as imagens estão em mesmo numero em cada classe e se a matriz dos pixeis tem o mesmo tamanho que o *array* do caminho das imagens;
- De vamos baralhar em uníssono o *array* das labels e a matriz das imagens. O objetivo é termos uma ordem diferente sempre que carregamos as imagens. Assim, cada treino é potencialmente diferente do outro.
- O próximo passo é a obtenção de uma matriz com os piteis de cada imagem por linha e que na ultima coluna tenha a *label* para ser utilizada na classificação;
- De seguida normalizamos o valor de cada pixel de cada imagem. A normalização é a divisão pelo valor máximo do pixel. Por exemplo, um pixel com valor de 125 é dividido por 255;
- O ultimo passo é aplicar *hot encoding*. Esta técnica permite reduzir a computação evitando calculas a mais entre vetores limitando o numero de cada índice dos mesmos à representação da sua classe.

O algoritmo do processo de criação da matriz da representação das imagens é o seguinte.

-
1. Criar lista de caminhos de imagens
 2. Para imagem na diretoria
 - a. Se a imagem não começar com “.”
 - i. Adicionar imagem à lista de imagens
 - b. Fim se
 3. Fim para
 4. Criar matriz para guarda a representação das imagens
 5. Para caminho na lista de caminhos de imagens
 - a. Se for para representação RGB
 - i. Imagem = ler imagem do caminho
 - b. Senão
 - i. Imagem = imagem convertida para escala de cinzentos
 - c. Fim se
 - d. Representação = Vetor de uma dimensão da imagem
 - e. Adicionar representação à matriz da representação das imagens
 6. Fim para
-

ALGORITMO 2 - CRIAÇÃO DE MATRIZ DE REPRESENTAÇÃO DE IMAGENS

De seguida são adicionadas as labels de cada imagem e criados os datasets de treino, validação e testes. O algoritmo do processo de criação dos datasets é o seguinte.

-
1. Labels de fogo e labels de floresta = 0
 2. Criar lista de labels com tamanho de lista de caminhos de imagens
 3. Para nomeImagem em lista de caminhos de imagens
 - a. Nome = separar nome pelo “-”
 - b. Se nome = “fire”
 - i. Adicionar o numero 1 na lista de labels com o índice igual ao enumerador do ciclo
 - c. Senão se nome = “floresta”
 - i. Adicionar o numero 0 na lista de labels com o índice igual ao enumerador do ciclo
 - d. Fim se
 4. Fim para
 5. Concatenar matriz de representação de imagens com vetor de labels e baralhar em uníssono
 6. Criar listas de tuples de imagen e label para dataset de treino, validação e teste. Dividir a matriz para os datasets correspondentes
 7. Converter as listas para arrays numpy e alterar a forma do array para(numero de items, numero de canais, tamanho vertical, tamanho horizontal)
 8. Dividir os valores de todos os pixéis de cada imagem pelo máximo de um pixel (255)
 9. Criar numpy arrays com valores por categoria $([0,0,0,0,1], [0,1,0,0,0])$
-

ALGORITMO 3 - CRIAÇÃO DOS DATASETS DE TREINO, VALIDAÇÃO E TEST

Parâmetros da rede

Depois de escolhida a arquitetura da rede, preparado o dataset, e carregadas as imagens, em matrizes prontas a serem consumidas pela *CNN*, foi necessário escolher vários parâmetros para otimizar a rede.

Um dos problemas que temos, é sem duvida alguma o dataset pequeno. Dependendo do problema, o dataset poderia ser suficiente. Mas neste caso não o é porque a imagem de fogo pode ser representada de várias formas e como tal gera várias ativações na rede, o que torna difícil treinar com um pequeno dataset. O dataset é o principal pela performance da rede. Depois do dataset vem a arquitetura que também ela já está escolhida. De seguida, o mais impactante são os parâmetros da rede. Nesta fase devido a limitações a nível de hardware a afinação foi pouca. Foram afinados os parâmetros de aprendizagem ou learning rate, optimizadores e configurados alguns *callbacks*.

Para a aprendizagem da rede, é utilizado o valor de 1.0. Esse valor, é um valor extremamente alto, mas que numa fase inicial ajuda a rede a não cair num mínimo local. Num cenário em que a rede está a aprender com o learning rate de 1.0, mas em que a perda no dataset de validação não diminui, foi configurado um *callback* para reduzir o learning rate com um factor de 0.2 até o valor mínimo de 0.0001. Este *callback*, só é utilizado se a condição se mantiver durante mais de quatro épocas. Esta configuração vem com um senão, se reduzimos muito o learning rate, temos de aumentar o numero de épocas a treinar.

Outro *callback* que foi configurado, foi a possibilidade de gravar a melhor época entre X épocas. Esta configuração permite ter a rede a correr por 300 épocas com learning rates muito baixos e guardar somente a época em que a perda no dataset de validação foi menor.

Por ultimo, a cada época é escrito para um ficheiro csv o valor de exatidão e perda nos datasets de treino e validação, bem como o valor de learning rate.

O optimizador utilizado, por ter melhor performance, foi o *rmsprop*. Este otimizador foi utilizado em detrimento do *stochastic gradient descent* e do *Adadelta*. Neste caso a escolha deveu-se apenas na performance em 10 épocas.

5.2.4 Serviço de streaming

O *RaspberryPi* é responsável por, alem da classificação das imagens o *streaming* das mesmas. Desta forma, o utilizador do sistema consegue confirmar eventuais situações de alarme ou até ter uma central de despacho para monitorizar uma área. O serviço assenta-se num servidor web. Sempre que o serviço é iniciado, é iniciado com ele um servidor web.

Depois do servidor web iniciado, é utilizado *MJPEG*(motion jpeg) para simular o *streaming* de vídeo. Na prática, estamos a enviar sucessivas imagens para o servidor web, que depois vão dar a sensação de movimento. Apesar de existirem outras formas de *streaming* de vídeo, esta é a forma que de uma maneira mais direta, permite modificar cada *frame*, passando-o pelo classificador e só depois enviando para o servidor web.

Este serviço está acessível pela internet com o seguinte endereço http://ip_publico_raspberry:5000. No contexto desta aplicação, o serviço será somente utilizado na aplicação móvel apesar de estar também disponível por qualquer browser.

O diagrama seguinte representa o funcionamento do serviço de *streaming*.

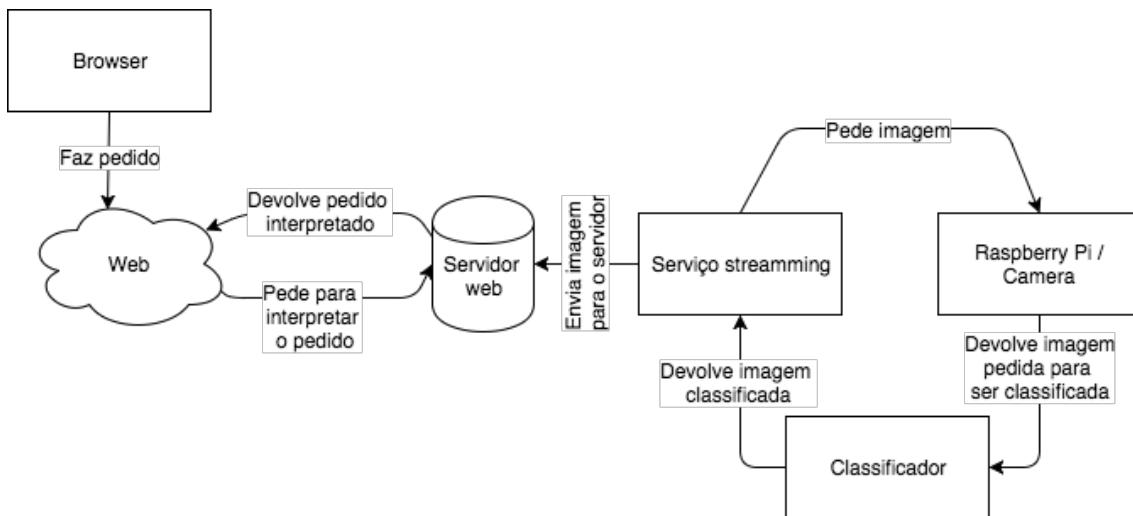


FIGURA 60 - DIAGRAMA FUNCIONAMENTO STREAMMING

5.2.5 Aplicação Móvel

No âmbito do projeto foi desenvolvida uma aplicação móvel tem três funcionalidades core:

- Login no sistema;
- *Streaming* de vídeo;
- Notificações remotas;
- Histórico de alteração dos estados do sistema;

A aplicação é uma aplicação *Ad-Hoc*. Ou seja, o objetivo não é ser distribuída através de uma *store* mas sim através da aquisição do sistema. Por essa mesma razão foi desenvolvida para *Android* uma aplicação simples que permite fazer login no sistema, visualizar o *streaming* bem como a sua classificação, e receber notificações remotas quando há alertas.

Login

Este módulo da aplicação permite de forma centralizada, o utilizador fazer login. A única razão da existência deste módulo é adicionar uma camada de segurança sobre quem acede às funcionalidades da aplicação.

O *Firebase* permite autenticação perante o serviço através de varias formas. Esta aplicação utiliza o método mais conhecido. Autenticação por e-mail e password.

O login é o primeiro era da aplicação. Quando instalada pela primeira vez, é mostrado o era ilustrado na seguinte imagem.

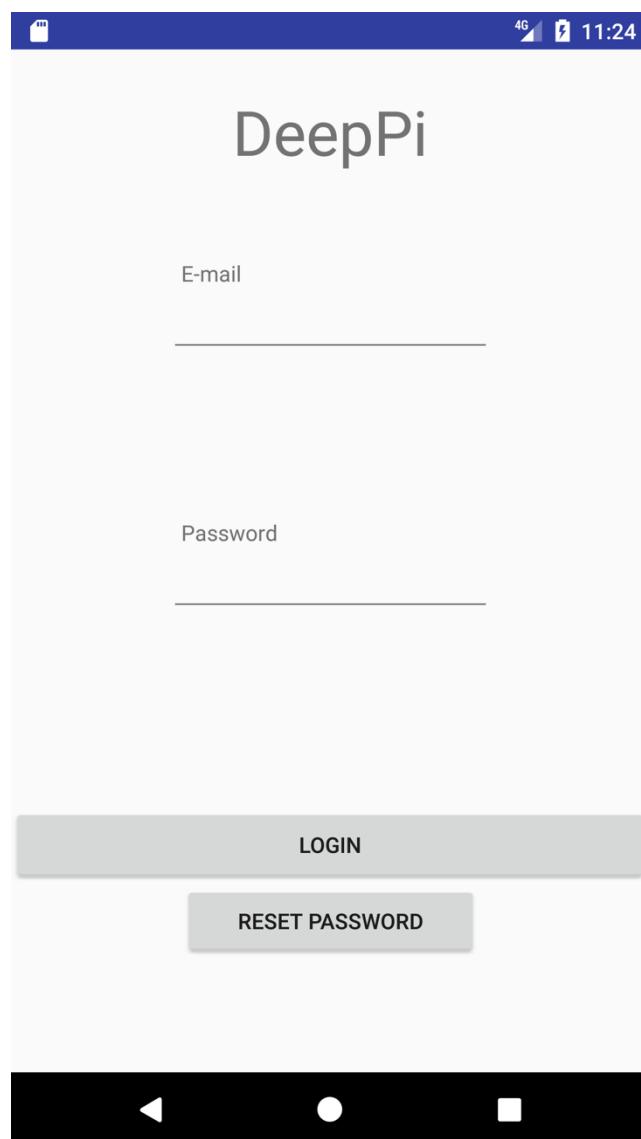


FIGURA 61 - ECRA DE LOGIN.

Também o e-mail carece de verificação. Este tem de estar corretamente formatado. A imagem seguinte mostra a tentativa de login com um e-mail mal formatado.

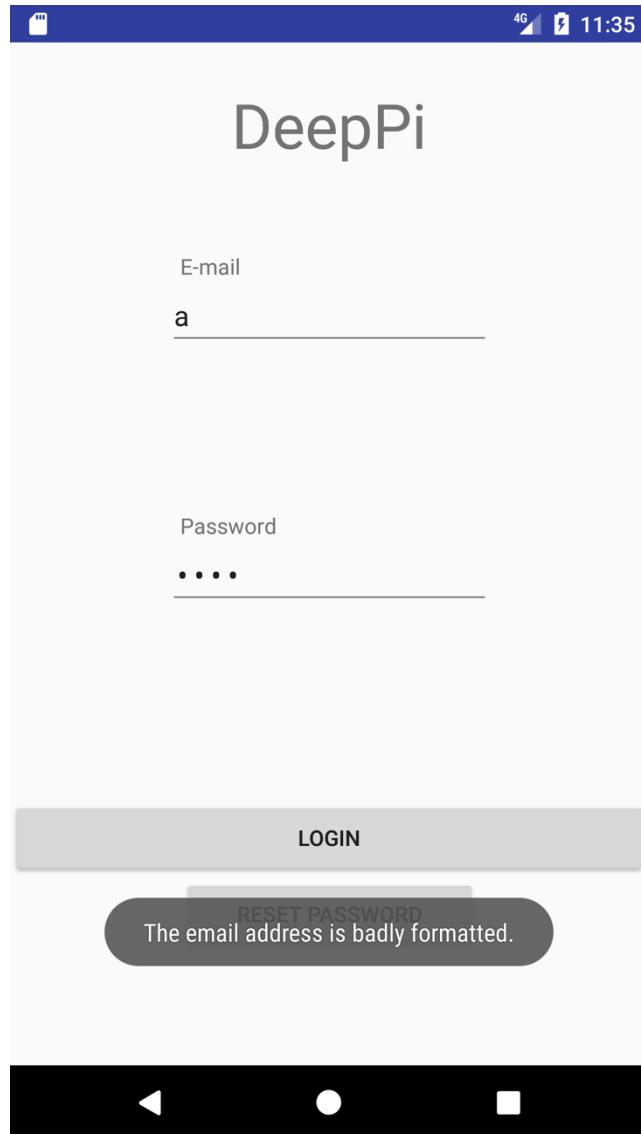


FIGURA 62 - EMAIL MAL FORMATADO.

Os campos obrigatórios têm validação para evitar chamadas desnecessárias. A seguinte imagem mostra a tentativa de login sem inserir credenciais.

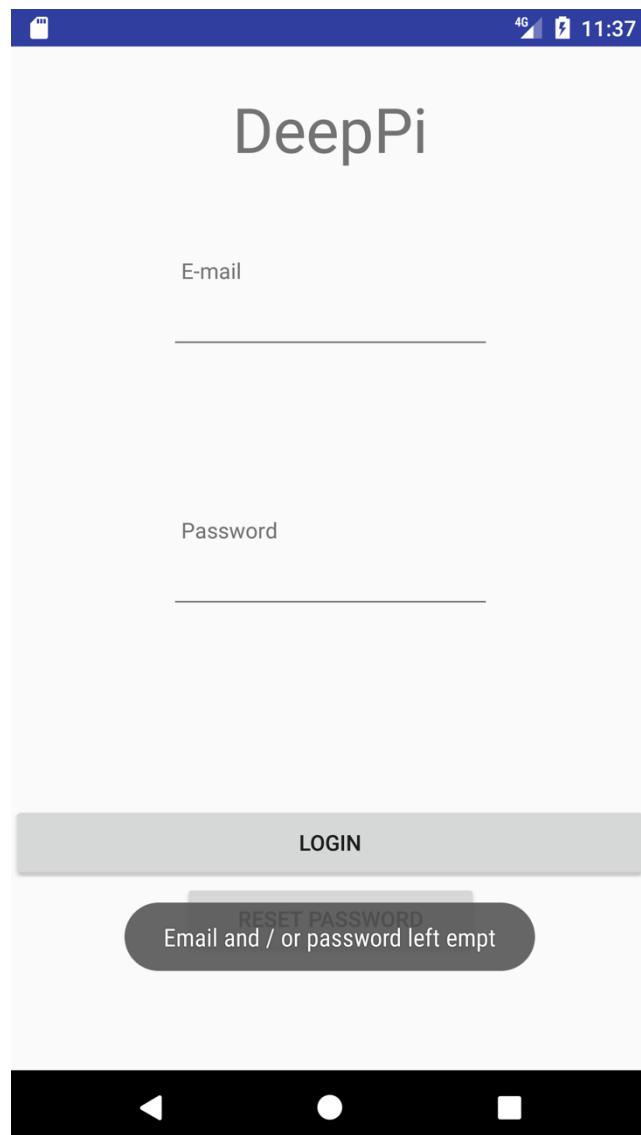


FIGURA 63 - CAMPOS VAZIOS.

No caso de as credenciais estarem certas, é mostrado um ecrã representado pela seguinte imagem, que significa que o pedido de autenticação está a ser feito.

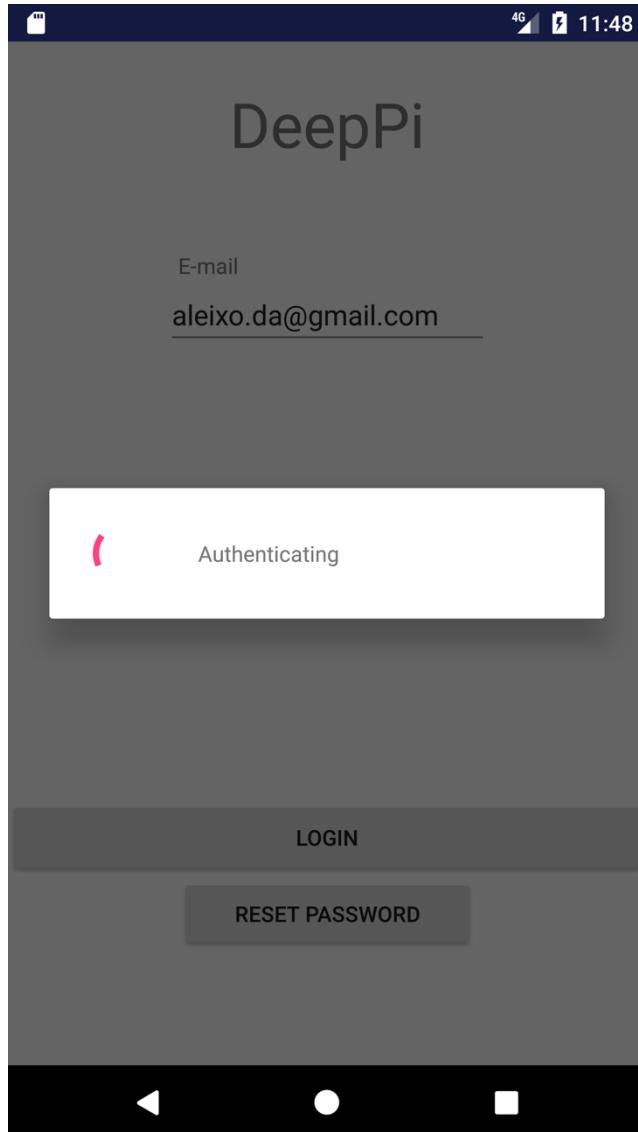


FIGURA 64 - CHAMADA DE AUTENTICAÇÃO.

Existe outra opção no contexto do login. O utilizador pode mudar a sua palavra passe. Devido a razões de segurança e proteção de dados sensíveis, o *Firebase* não permite de forma direta mudar a palavra passe de um utilizador. O utilizador que quer mudar a palavra passe tem de ter feito o login pelo menos uma vez o que obriga a avisar o utilizador quando tenta mudar uma password sem ter feito login como mostra a imagem.

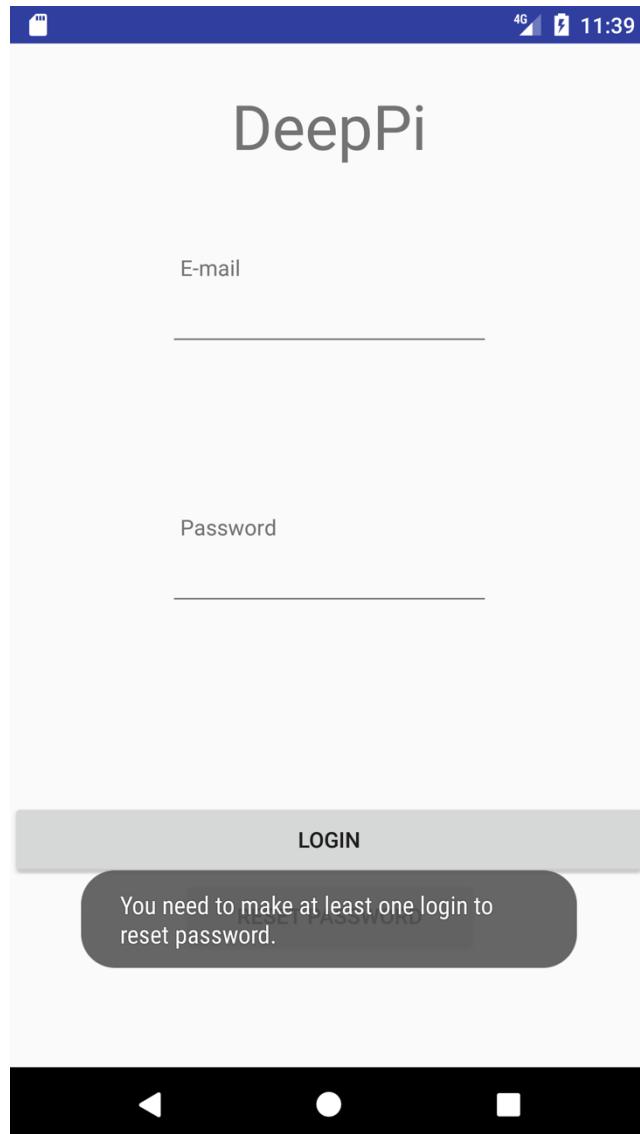


FIGURA 65 - TENTATIVA DE RESET DA PASSWORD FALHADA.

Caso já tenha sido feito alguma vez o login é mostrado um alerta para definir a nova palavra passe como mostra a seguinte imagem.

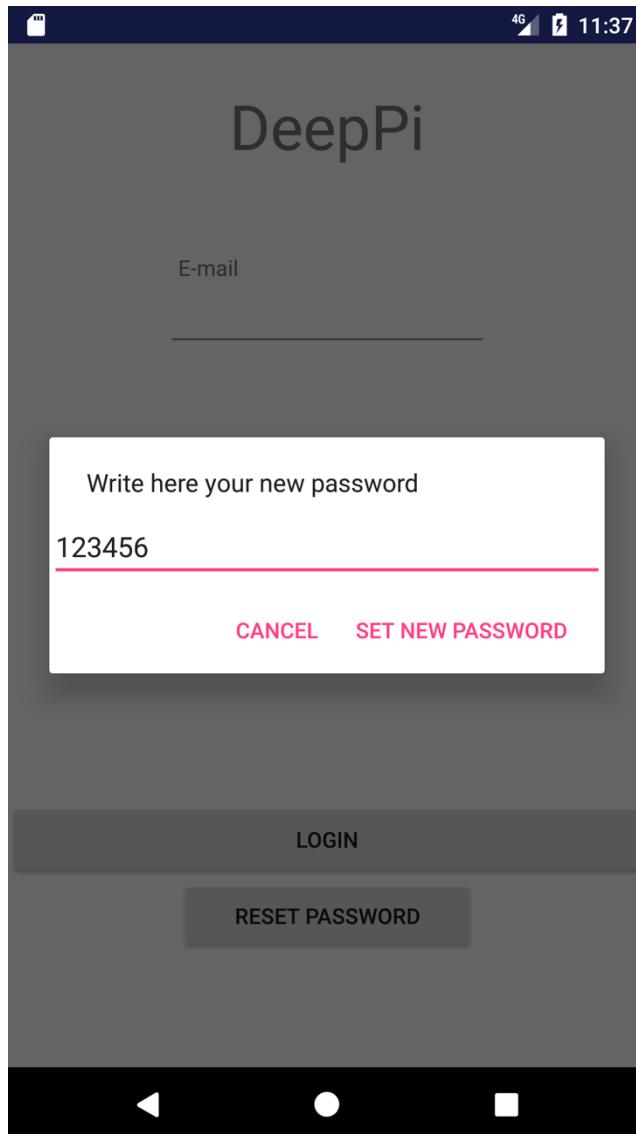


FIGURA 66 - ECRA PARA INSERIR NOVA PASSWORD.

Depois de fazer “SET NEW PASSWORD”, é feita uma chamada com o pedido de alteração da password. O sucesso da chamada é representado na próxima imagem.

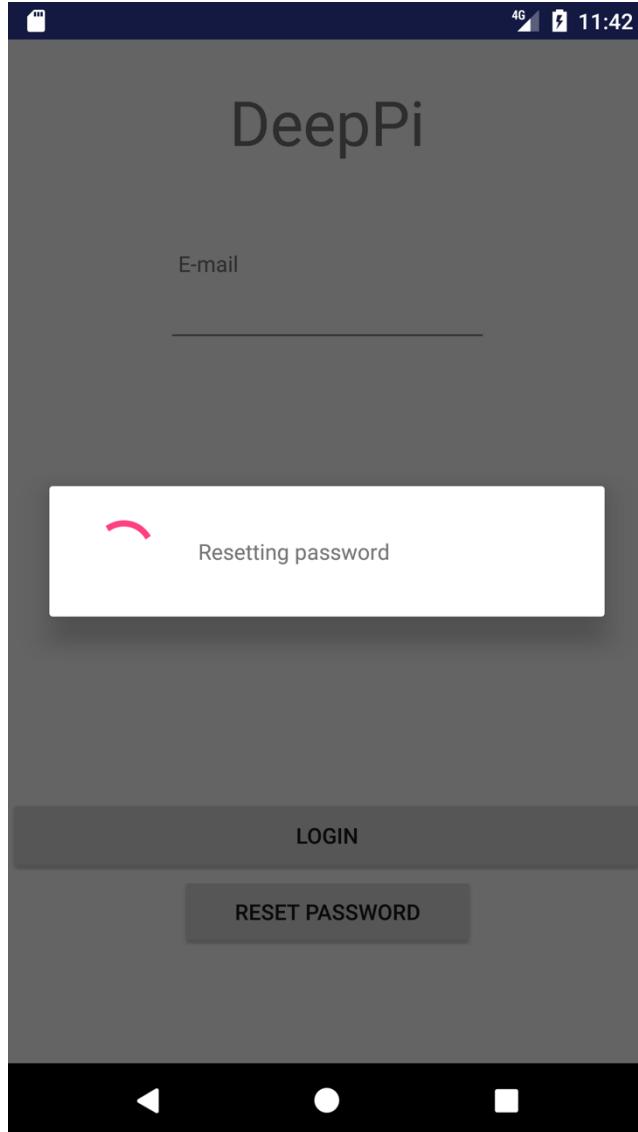


FIGURA 67 - CHAMADA PARA RESET DA PASSWORD.

No caso de existir algum erro, o componente de espera é retirado e é mostrada uma mensagem ao utilizador. Caso seja bem-sucedido, o login é feito como demonstra a seguinte imagem.

Streaming

Após login, o utilizador tem disponível o *streaming* do vídeo recolhido pelo sistema. O vídeo é transmitido em tempo real bem como a classificação de cada *frame*. A seguinte imagem mostra o *streaming* de vídeo.

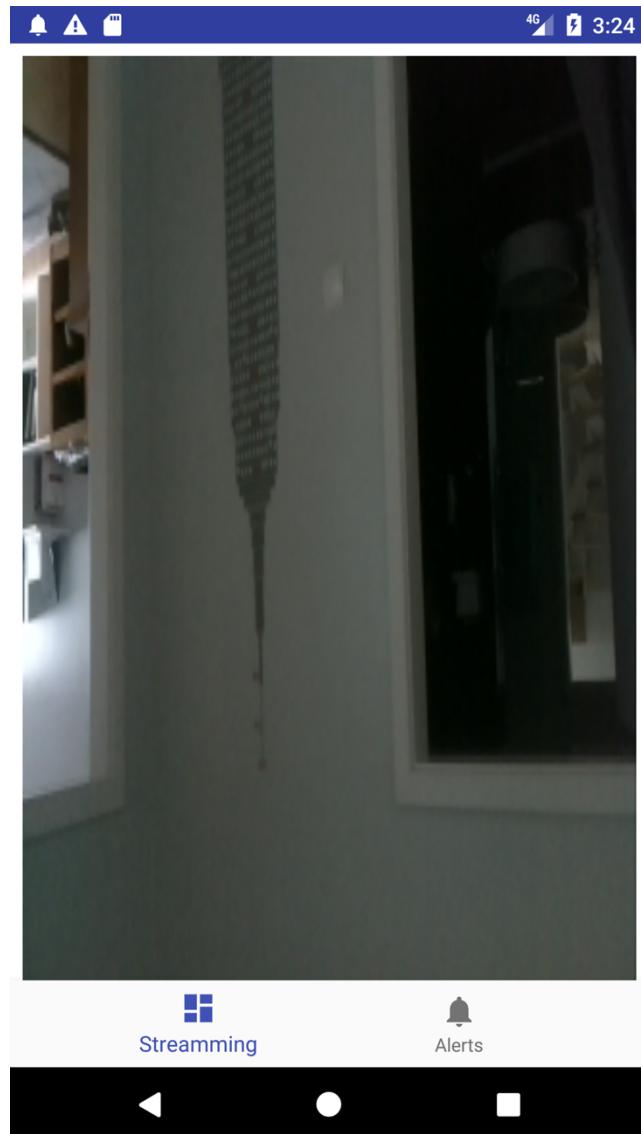


FIGURA 68 - ECRA DE STREAMMING DO VÍDEO.

Quando o utilizador recebe um alarme a indicar fogo, consegue validar através do *streaming* do vídeo o estado do local observado.

Notificações

As notificações informam o utilizador da alteração da classificação do que a camera está a ver. Na prática, o utilizador só recebe uma notificação remota quando o sistema deteta fogo ou existe um modelo de treino melhorado para fazer download.

As notificações são enviadas e recebidas em qualquer telefone que tenha a aplicação instalada e mostradas em qualquer estado a aplicação. A seguinte imagem mostra uma notificação remota em *foreground*.

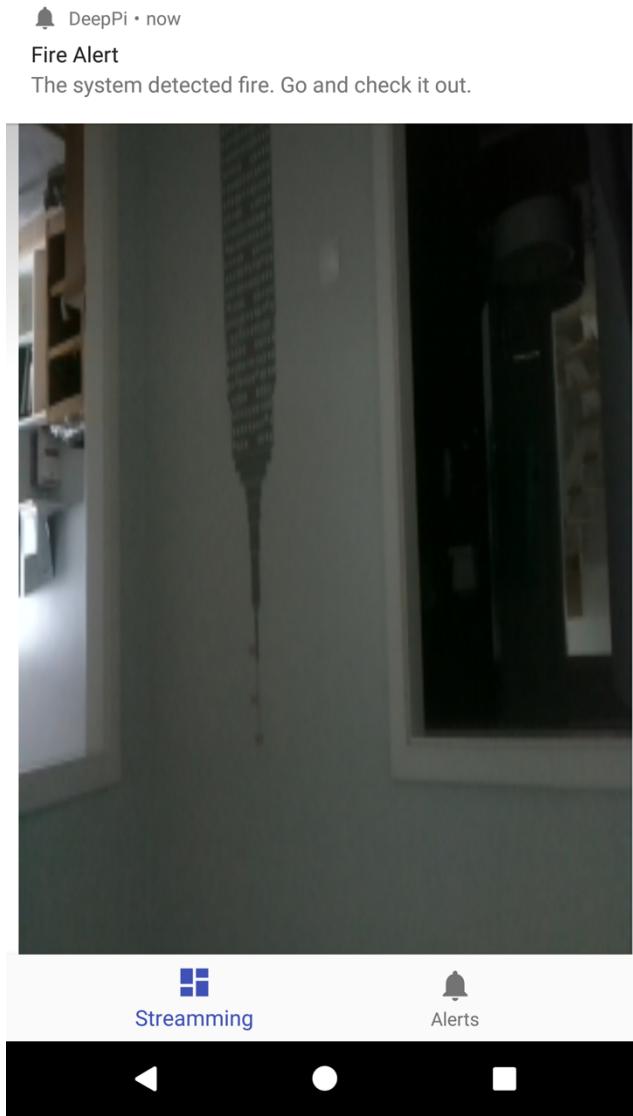


FIGURA 69 - NOTIFICAÇÃO PUSH EM FOREGROUND.

A seguinte imagem mostra uma notificação remota em *background* em qualquer estado a aplicação.

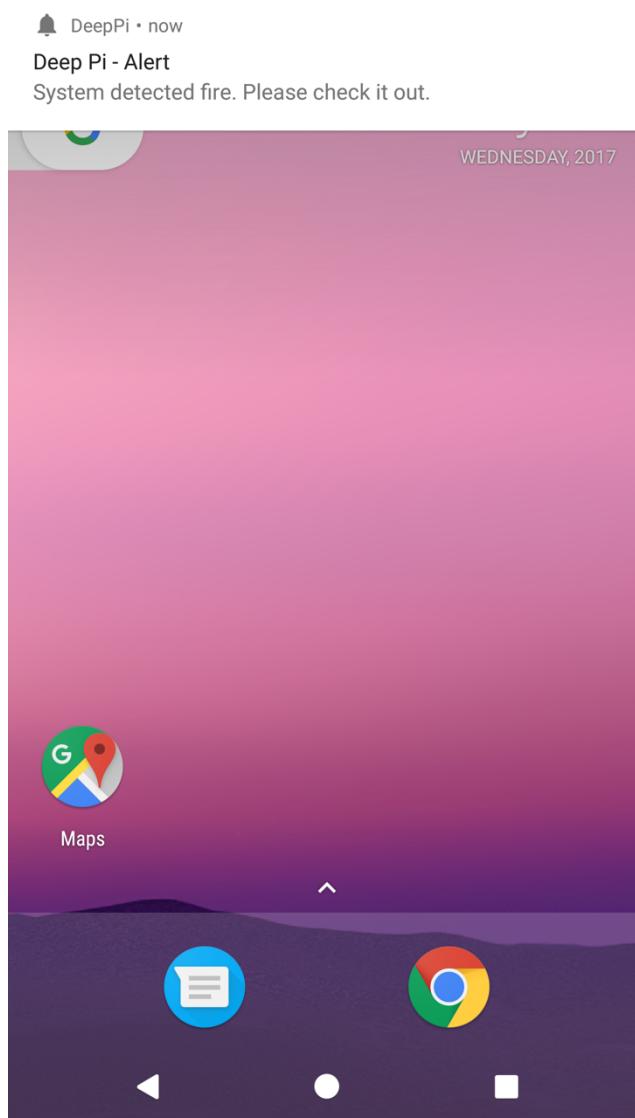


FIGURA 70 - NOTIFICAÇÃO PUSH EM BACKGROUND.

Histórico

Este modulo da aplicação permite ao utilizador saber todos os alertas que foram recebidos.

Este módulo utiliza uma base de dados que utiliza todas as aplicações cliente em tempo real. Na prática, faz uma *query* a uma base de dados no *Firebase* e depois fica à escuta de eventos nessa mesma base de dados. Cada vez que existe um evento, atualiza uma lista de alertas dentro da aplicação.

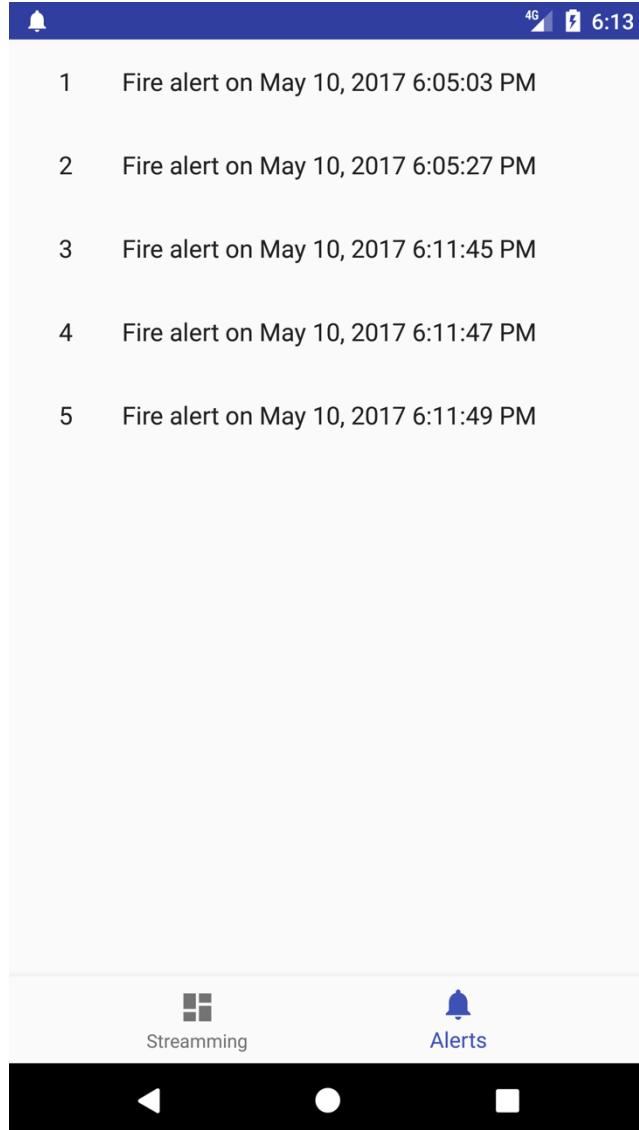


FIGURA 71 - HISTÓRICO DAS NOTIFICAÇÕES DE FOGO.

5.2.6 Módulo Pan&Tilt

De modo a conseguir uma rotação da câmara, foi adicionado ao sistema um modulo de *pan&tilt*. O objetivo deste modulo é adicionar ao sistema a capacidade de rotação da câmara no eixo do *X* e do *Y*.

Numa primeira analise foi desenvolvido um script em *Python* para o efeito. O script utiliza um *wrapper* para python chamado *RPi.GPIO* que permite de forma fácil aceder ao *GPIO* (General pin input output). Alem da facilidade que dá ao programador, o *RaspberryPi 3* tem como lacuna o facto de ter apenas uma porta com hardware para lidar com *PWM* (Pulse width modulation). O *wrapper*, possibilita adicionar a capacidade de lidar com *PWM* a qualquer porta do *GPIO*. Depois de desenvolvido o script, foi verificado

que o servo está sempre a ajustar a sua posição mesmo sem ser enviado nenhum pulso. Tal facto é justificado com o *RaspberryPi* não ter um *kernel* de tempo real. O *Raspbian*, *SO* (Sistema operativo) utilizado, é baseado em *linux*. O problema que aqui se põe e a razão dos constantes acertos do servo, é que o *kernel* dá prioridade a todas a quaisquer tarefas que o sistema operativo tem para fazer, o que resulta em variações do pulso.

Alem do problema mencionado, o *RaspberryPi* por si só não é um sistema muito poderoso a nível de hardware. Lançar um processo para iniciar e controlar os servos pode ser delegado para hardware que possa fazer somente isso. Neste caso, foi utilizado o Arduino nano para controlar os servos. A ligação dos servos é demonstrada na seguinte imagem.

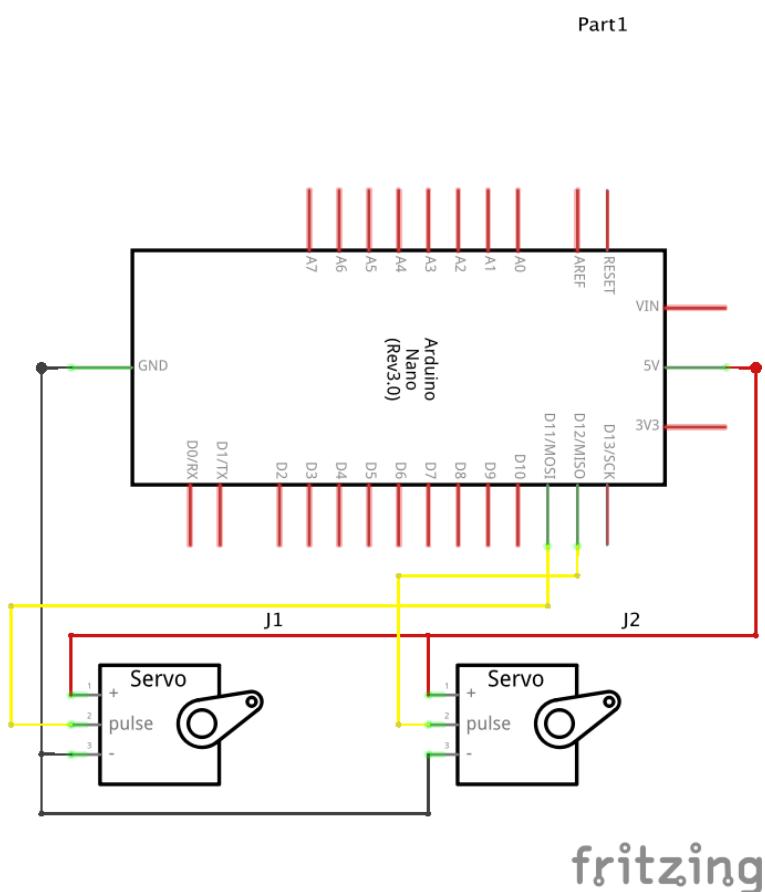


FIGURA 72 - ESQUEMATICO LIGAÇÃO SERVO MOTORES.

5.2.7 Módulo de energias renováveis

O sistema está dotado de um modulo que permite ser suficiente autossuficiente energeticamente. Para ser projetado foi necessário primeiro ver qual a corrente máxima

consumida pelo *Raspberry Pi 3* quando utiliza os quatro núcleos do processador. A seguinte imagem mostra um gráfico da corrente utilizada por todos os *Raspberry Pi*.

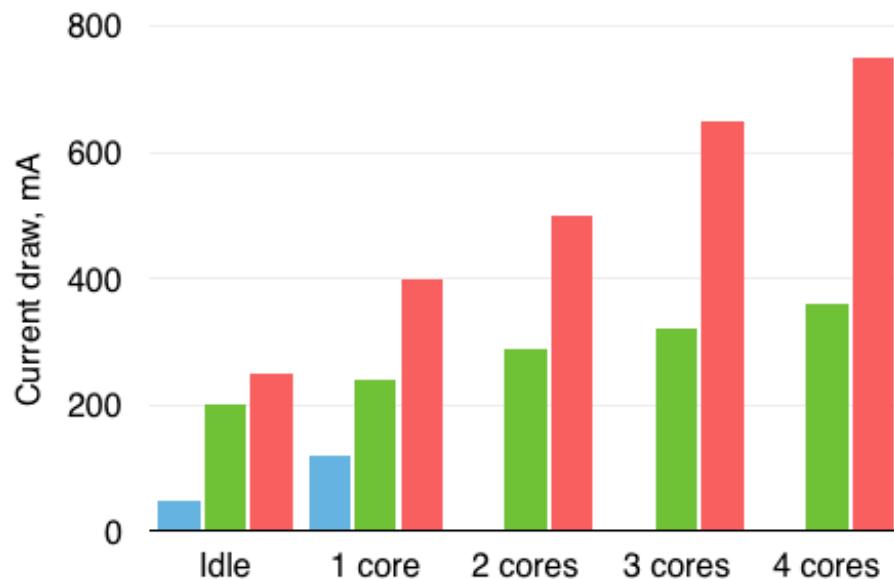


FIGURA 73 - CONSUMO FAMILIA RASPBERRY PI. ADAPTADA DE [11]

De modo a termos uma margem de erro, para o módulo assumimos que o hardware vai consumir 1 Ampere.

6 Testes e resultados

Neste capítulo são mostrados os testes feitos tanto ao classificador como ao sistema como um todo. Estes testes têm a capacidade de validar o bom funcionamento do sistema.

6.1 Testes do classificador

Num classificado deste tipo, o mais difícil é a obtenção da arquitetura certa com os parâmetros certos. Existem pela comunidade científica várias arquiteturas de *CNN's* bastante estudadas e utilizadas. As arquiteturas mais conhecidas são, a *GoogleNet*, *lenNet*, *AlexNet* entre outras. Em torno do tema das arquiteturas existe bastante investigação que é feita e até concursos como o *kaggle* que disponibiliza vários dataset e o objetivo é ter a maior eficácia. No âmbito deste projeto foram estudadas várias arquiteturas conhecidas, desenvolvidas duas e foram todas comparadas com os seguintes pressupostos.

- O treino é feito numa versão pequena do dataset;
- Os *hyperparameters* da rede não são otimizados;
- O treino é feito em 10 épocas;
- 20% do dataset é utilizado para validação
- 20% do resultado do passo anterior é utilizado para validação

No *ambito* de testes da arquitetura ideal para o classificador, foram testadas quatro arquiteturas.

A primeira arquitetura a ser utilizada, e a mais simples de todas foi a *LeNet*. Esta é uma arquitetura conhecida pela comunidade científica. A seguinte imagem mostra a arquitetura *LeNet*.

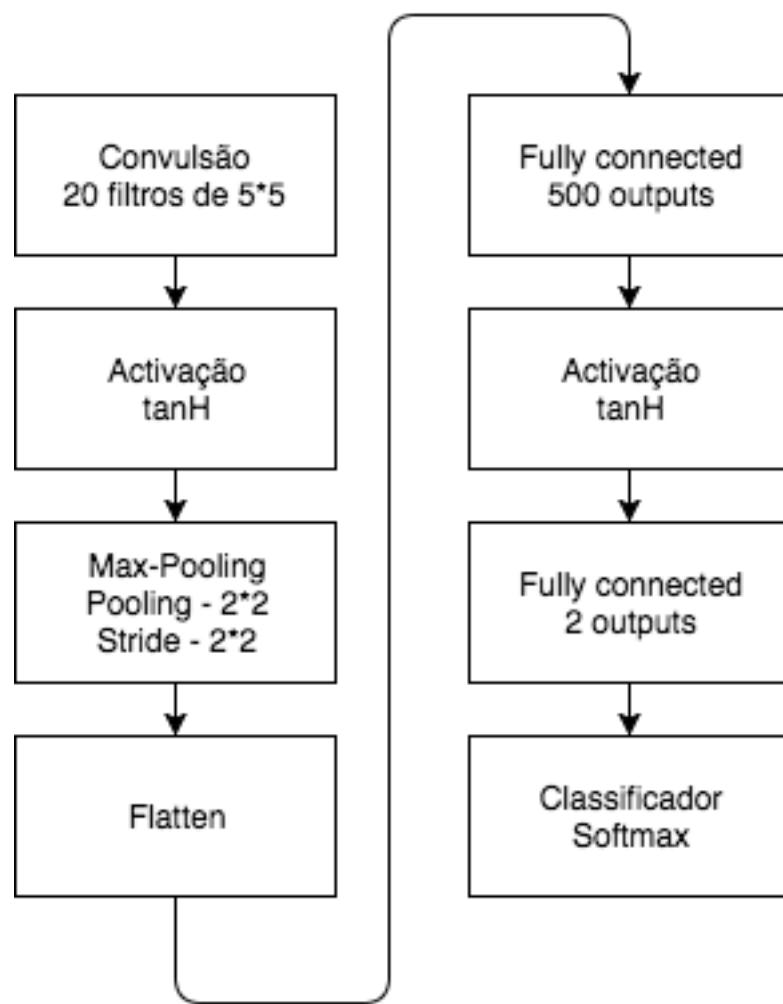


FIGURA 74 - ARQUITETURA LENET.

A próxima arquitetura, baseada na arquitetura *LeNet* e em conceitos teóricos, foi uma arquitetura por mim desenvolvida.

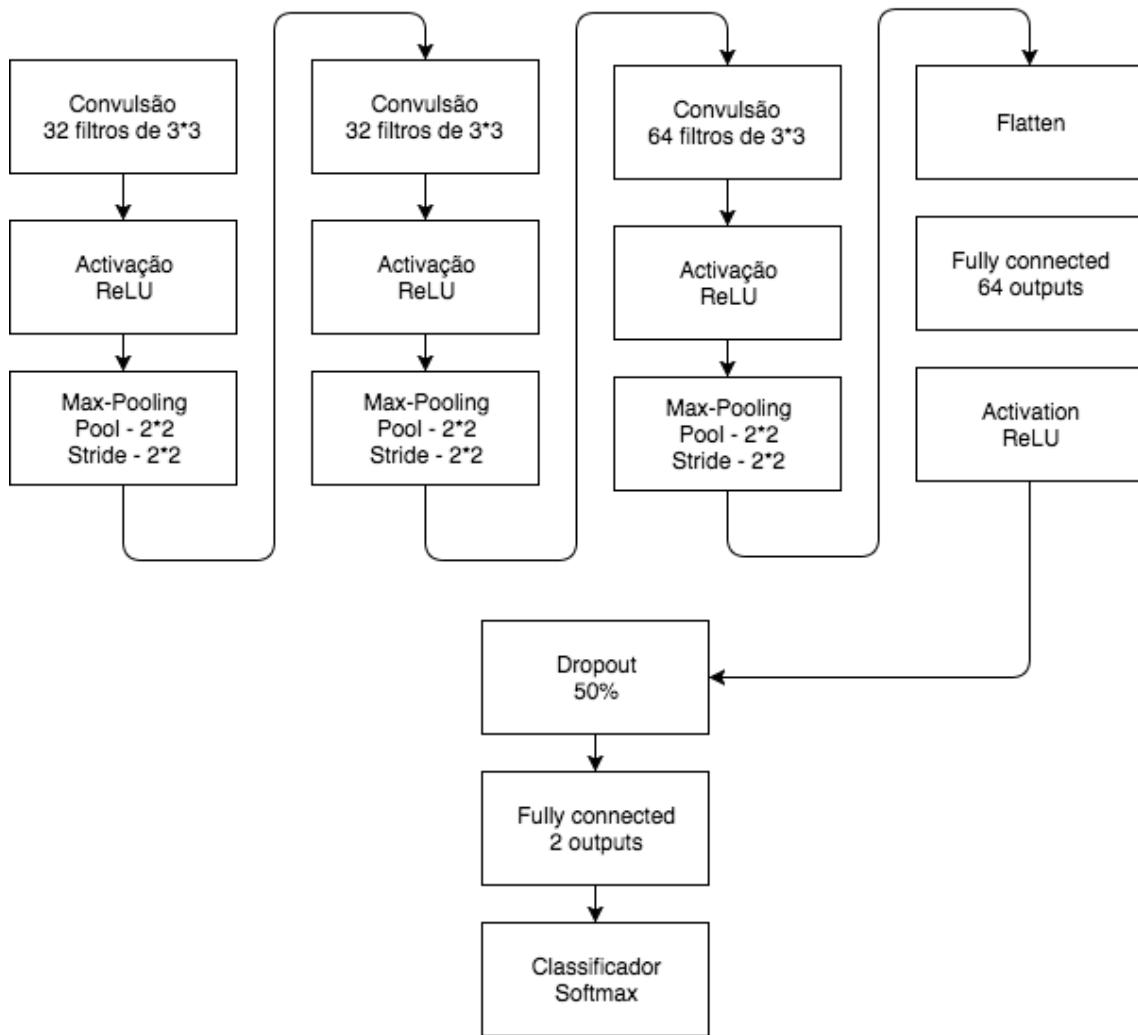


FIGURA 75 - ARQUITETURA PERSONALIZADA.

A próxima arquitetura a ser estudada foi desenvolvida por um estudante de *Standford* e tem o objetivo ser uma arquitetura muito “leve”. Esta arquitetura é tão leve que é capaz de correr num browser web através de *javascript*. A seguinte imagem mostra a arquitetura de *Karpathy*.

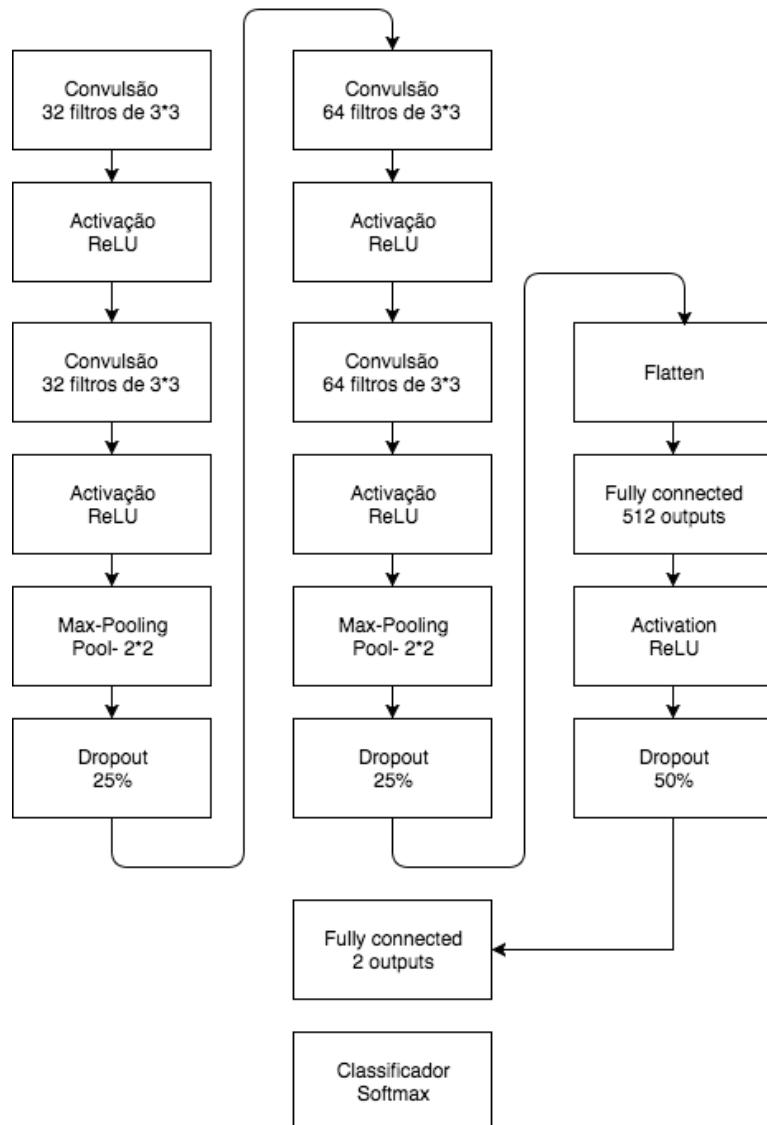


FIGURA 76 - ARQUITETURA KARPATHY.

A ultima arquitetura analisada foi uma adaptação da arquitetura *VGGNet*. A *VGGNet*, é uma arquitetura muito conhecida mas muito pesada, pelo que é preciso um *GPU* (Graphic processing unit) para treinar em cima da mesma. Neste caso, e devido a limitações de hardware, foi decidido utilizar uma versão simplificada da *VGGNet*. A seguinte imagem mostra a arquitetura.

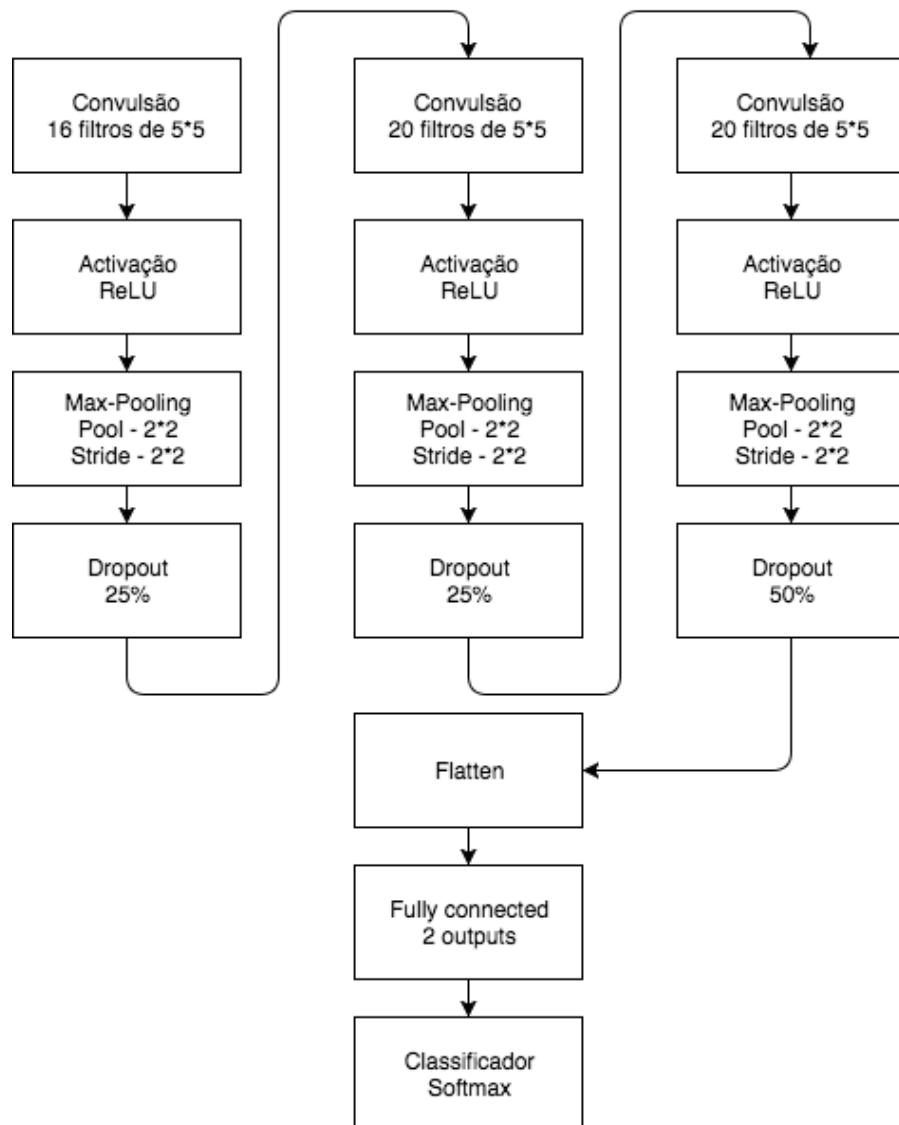


FIGURA 77 - ARQUITETURA MINI VGG.

Arquitetura final

A arquitetura final foi a versão simplificada da *VGGNet*. No âmbito deste projeto, devido a restrições a nível de hardware não foi possível afinal os parâmetros da rede para além do afinado na fase de testes de arquitetura.

A fase final contou com treino em 60 épocas e um dataset de 18088 imagens. O treino foi feito com base nas imagens em escala de cinzentos e conseguiu as seguintes métricas:

- Perda no dataset de treino - 0.28;

- Perda no dataset de validação - 0.26;
- Exatidão no dataset de treino - 90%;
- Exatidão no dataset de validação - 91%;

Estas métricas sugerem um dataset pequeno pois o sistema consegue classificar melhor o dataset de validação que o dataset de treino.

	<i>Previsão negativa</i>	<i>Previsão positiva</i>	<i>Total</i>
<i>Exemplo negativo</i>	1712	100	1812
<i>Exemplo positivo</i>	213	1593	1806

TABELA 1 - MATRIZ DE CONFUSÃO.

Existem duas possibilidades neste dataset. Ou é floresta (negativo) ou é fogo(positivo). Na tabela anterior, é mostrada a relação entre previsões positivas e negativas. Podemos analisar a percentagem de vezes que o classificador está correto e errado para cada classe:

- **Verdadeiros negativos** - Prevemos que não era fogo e não era;

$$\frac{1712}{1812} * 100 = 95\%$$

- **Falsos positivos** - Prevemos que era fogo e não era;

$$\frac{100}{1812} * 100 = 5\%$$

- **Falsos negativos** - Prevemos que era fogo e não era;

$$\frac{213}{1806} * 100 = 11\%$$

- **Verdadeiros positivos** - Prevemos que era fogo e era;

$$\frac{1592}{1806} * 100 = 89\%$$

Existem várias métricas extra importantes de referir. A exatidão do sistema é uma métrica que nos diz a quantidade de vezes em que o valor está corretamente classificado (Verdadeiros negativos e verdadeiros positivos). Para calcular a exatidão utilizamos a seguinte equação.

$$\frac{(Vp + Vn)}{len} * 100$$

Em que Vp são os verdadeiros positivos, Vn são os verdadeiros negativos e len é o tamanho do dataset.

$$\frac{(1592 + 1712)}{3618} * 100 = 91,3\%$$

Outra métrica importante é o *Recall*. O *Recall* diz quando é verdade, quantas vezes o classificador classifica como verdade.

$$\frac{Vp}{len} * 100$$

Em que Vp são os verdadeiros positivos e len é o tamanho do dataset.

$$\frac{1593}{1806} * 100 = 88\%$$

Podemos também ver quantas vezes no total está o classificador errado com a seguinte equação:

$$\frac{Fp + Fn}{len} * 100$$

Em que Fp são os falsos positivos, Fn são os falsos negativos e len é o tamanho do dataset.

$$\frac{100 + 213}{3618} * 100 = 8,7\%$$

Outra métrica interessante é a precisão do classificador. Quando prevê fogo com que frequência está correto. A equação utilizada é a seguinte:

$$\frac{Vp}{Vp + Fp} * 100$$

Em que Vp são os verdadeiros positivos e Fp são os falsos positivos.

$$\frac{1593}{1593 + 100} * 100 = 94\%$$

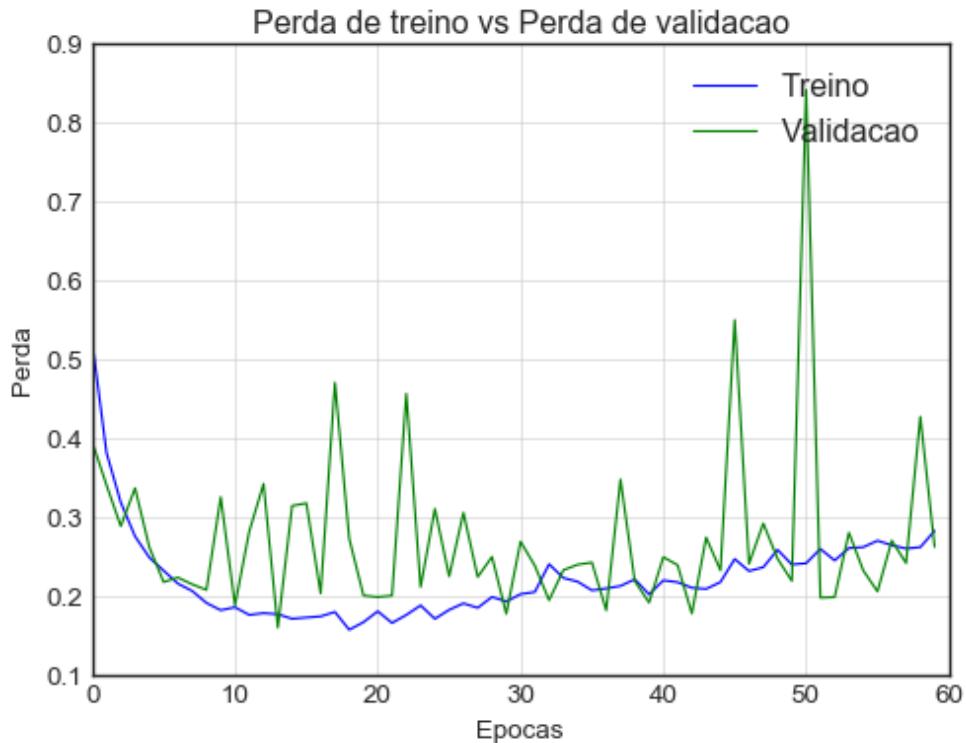
Existem ainda outras métricas, mas que para este exemplo não são necessárias para validar a performance do sistema. A seguinte tabela apresenta a precisão e recall para cada uma das classes, bom como uma média das métricas do sistema.

	Precisão	Recall
Floresta	89%	94%
Fogo	94%	88%

<i>Média</i>	92%	91%
--------------	-----	-----

TABELA 2 - PRECISÃO, RECALL E F-SCORE.

Depois de obter os valores da matriz de confusão e interpretar os essenciais, é bom para quem está a otimizar o classificador ter uma visão geral do treino. Nesse sentido, a seguinte imagem mostra a diferença e evolução da perda no dataset de treino e de validação.

**FIGURA 78 - PERDA EM TREINO E FASE DE VALIDAÇÃO.**

A perda dá uma ideia de como a rede está a aprender e o quanto os valores dos pesos e bias atuais divergem do que seria ideal. Outro gráfico importante é a exatidão do classificador enquanto treino e validação. A seguinte imagem dá uma visão geral da exatidão do sistema.

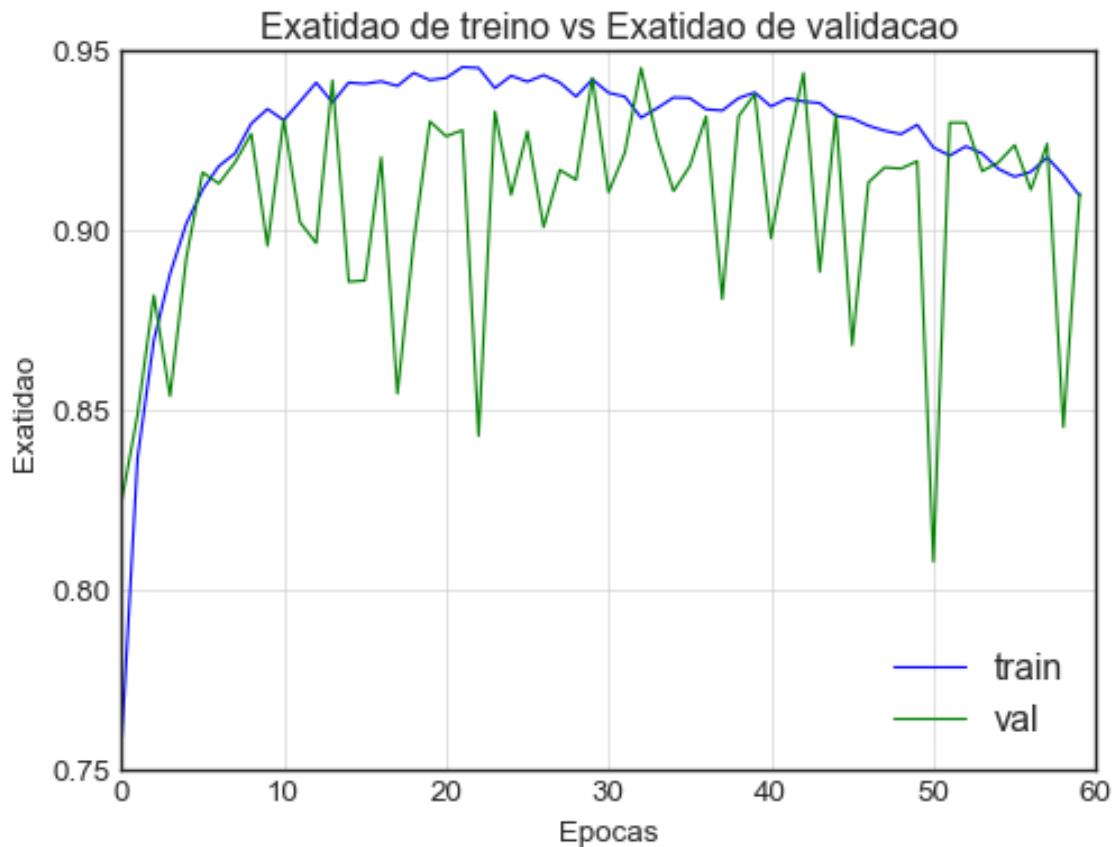


FIGURA 79 - EXATIDÃO EM TREINO E FASE DE VALIDAÇÃO.

Podemos ver com a anterior imagem a evolução do classificador em 60 épocas. Através desta visão de alto nível da perda e exatidão concluímos que o dataset de validação é pequeno e o classificador não consegue convergir bem. Esta conclusão é devido aos picos no dataset de validação. Apesar de não ser critico, pois, a variação é pouca, é sem duvida um aspeto a melhorar.

Por fim, e como prova de conceito, podemos ver algumas previsões na seguinte imagem. Foi utilizado o classificador treinado para num conjunto de 3618 imagens nunca antes vistas até ter sido construída a matriz de confusão, classificar 25. No eixo do Y está a previsão, e no eixo do X a classificação por mim dada a determinada imagem.

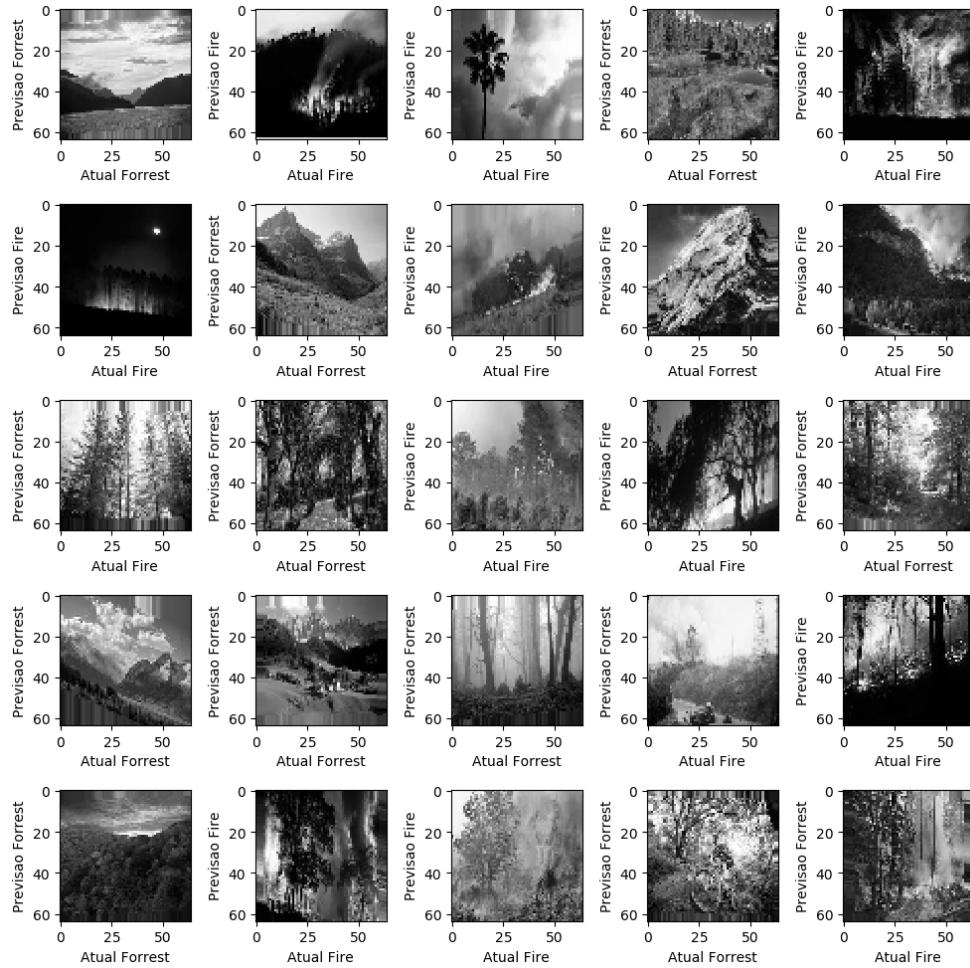


FIGURA 80 - PREVISÕES NO DATASSTE DE TESTE.

Podemos também observar os filtros gerados pelas camadas. A próxima imagem mostra os filtros gerados pela primeira camada convolucional.

Após as conclusões chegadas nesta secção, alterou-se o tamanho das imagens de 64 para 128 e o dataset de validação passou a ser 30% do dataset total. Também o *learning rate* passou a sofrer uma alteração com um factor de 0.5 por cada duas épocas que a perda da validação não diminua. Estas alterações levaram a uma exatidão o sistema de sensivelmente 98%.

FALTAM OS GRAFICOS DA EXATIDÃO E PERDA DA REDE

FALTA A MATRIZ DE CONFUSÃO

A REDE VAI NA EPOCA 40 DE 60. AINDA DEVE FALTAR QUASE UM DIA... DE QUALQUER MANEIRA, JÁ TENHO OS PESOS PARA UMA EPOCA COM 97,7% DE EXATIDÃO

Testes de sistema

Os testes efetuados foram bastante limitativos no que toca a validação da deteção de incêndio.

Idealmente, o sistema seria colocado numa situação real onde aprende exatamente sobre a localização onde se encontra para conseguir detetar mudanças e classifica-las como sendo ou não fogo.

No âmbito deste projeto, o maior teste foi a validação da possibilidade de correr uma CNN relativamente complexa num sistema de baixo custo como é o *RaspberryPi 3*. Seguem-se os planos de teste feitos e resultados.

- Correr o script para treinar uma rede:

- Fez o download de novas imagens de fogo com as *queries* definidas;
- Eliminou imagens duplicadas;
- Redimensionou imagens;
- Aumentou o dataset com recurso a técnicas de *augmented image*;
- A rede foi treinada;
- Foi feito o *upload* do ficheiro de arquitetura (.json) para o *Google drive*;
- Foi feito o *upload* do ficheiro dos pesos e *biases* da rede (.h5) para o *Google drive*:
- Foi enviada uma notificação remota a indicar que existe um novo treino disponível;

- Correr o script de teste:

- Fez o download do ficheiro de arquitetura do *Google drive*;
- Fez o download do ficheiro de pesos e *biases* do *Google drive*;
- Classificou uma imagem;
- Enviou uma notificação quando detetou fogo;

- Teste de stress às notificações:
 - Numa situação de fogo e não fogo constante só é enviada uma notificação de 30 em 30 segundos.
- Teste com vídeos do youtube:
 - Os vídeos do youtube são classificados corretamente tendo em conta as restrições.
 - O sistema ainda não generaliza bem para o nevoeiro e nascer do sol;

7 Conclusões

No geral, o projeto foi bastante gratificante em vários níveis. O primeiro foi obviamente a aprendizagem. Além da linguagem de programação *python*, programação em *Android* e tecnologias utilizadas que não foram abordadas em contexto académico, a maior aprendizagem de um tema completamente novo para mim foi na área de *machine learning* e visão computacional.

Conseguimos concluir após a elaboração dos testes, que é possível classificar com um bom grau de exatidão fogo de não fogo, e que é possível fazê-lo em dispositivos de baixo custo como é o caso do *RaspberryPi*.

Em relação à contribuição para o estado da arte do projeto, o mesmo vem acrescentar ao que já existe, uma solução baseada em tecnologias emergentes como o caso das *CNN's* o que a nível de é sem dúvida vantajoso pois o sistema vai ser capaz de se adaptar a uma situação em concreto.

Apesar do sistema ter ficado a funcionar, o mesmo carece de várias melhorias para serem efetuadas no futuro. Essas melhorias não foram feitas essencialmente porque seria preciso investigação também noutras áreas o que não se consegue encaixar no tempo da tese. Também o facto do sistema ter sido desenvolvido e treinado com *CPU* e não com *GPU* limitou muito o experimentar novas soluções. Os futuros passos para melhoria do sistema são:

- Recolha de imagens para o dataset;
- Testes com vários parâmetros de rede e combinações diferentes de arquiteturas;
- Tratamento das imagens;
- Investigação para afinação dos parâmetros da rede.
- Aplicação desktop;
- Testes em ambiente real;

Bibliografia

- [1] - Michael, N. (2017). *Neural networks and Deep Learning*. Retrieved from <http://neuralnetworksanddeeplearning.com/>
- [2] - Adrian, R. (2015.). PyImageSearch.
- [3] - Udacity. (2017). Self driving car nanodegree.
- [4] - Fire, Z. (n.d.). Zero Fire Systems.
- [5] - FireWatch. (n.d.). FireWatch - Early Detection of Forest Fires for Large Areas.
- [6] - N, B. K., & , Kwang-Ho Cheong, J.-Y. N. (2010). Early fire detection algorithm based on irregular patterns of flames and hierarchical Bayesian Networks. *ELSEVIER*.
- [7] - Qiang Yan, Pei Bo, e Z. J. (2014). Forest Fire Image Intelligent Recognition based on the Neural Network. *Journal of Multimedia*, 9.
- [8] - Armando M. Fernandes, Andrei B. Utkin, Alexander V. Lavrov, R. M. V. (2004). Development of neural network committee machines for automatic forest fire detection using lidar. *Journal Pattern Recognition*, 37(10), 2039–2047.
- [9] - Zhang, D., Han, S., Zhao, J., Zhang, Z., Qu, C., Ke, Y., & Chen, X. (2009). Image Based Forest Fire Detection Using Dynamic Characteristics With Artificial Neural Networks. *International Joint Conference on Artificial Intelligence*.
- [10] - Byoung Chul Ko, Kwang-Ho Cheong, J.-Y. N. (2009). Fire detection based on vision sensor and support vector machines. *ELSEVIER*, 44(3), 322–329.
- [11] - Raspberry Pi consumption. (n.d.).